TAMPERE UNIVERSITY OF TECHNOLOGY

Tuomas Tikkanen

CELL DETECTION FROM MICROSCOPE IMAGES USING HISTOGRAM OF ORIENTED GRADIENTS

Master of Science Thesis

Examiners:
University Lecturer Heikki Huttunen
Dr. Pekka Ruusuvuori

Examiners and topic approved by the
Council of the Faculty of Computing and
Electrical Engineering on 13 August 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing
TIKKANEN, TUOMAS: Cell Detection from Microscope Images Using Histogram of Oriented Gradients
Master of Science Thesis, 50 pages
November 2014
Major: Computational Systems Biology
Examiners: University Lecturer Heikki Huttunen & Dr. Pekka Ruusuvuori
Keywords: Cell Detection, Histogram of Oriented Gradients, Growth Curve, Machine Learning, Support Vector Machine, Object Recognition, Image Processing

In this master's thesis, cell detection from bright-field microscope images is studied using Histogram of Oriented Gradients (HOG) features with Support Vector Machine supervised learning classifier. The proposed method is trained iteratively by finding hard examples. The performance of the method is evaluated using 16 training and 12 test images with altogether 10736 PC3 human prostate cancer cells. The cell detection accuracy is assessed with Receiver Operating Characteristic curves, F_1-score and Bivariate Similarity Index. Decision between true positive and false positive detection is made using PAS-metric.

The experiments consider various parameters and their effect on performance. It is shown that using hard examples in the training phase increases the level of generalization of the model considerably. ROC AUC reaches an excellent value of 0.98 after iterative training. When SVM threshold is varied for each image in the testing phase, F_1-score averaged over the peak F_1-scores of each image reaches a high value of 0.85. The most suitable combinations of HOG descriptor parameter values are presented. All in all, results indicate that HOG can be successfully applied to bright-field microscope images of PC3 prostate cancer cells taken on subsequent days, which results in a growth curve that favorably agrees with manual counts. The implemented cell detection framework outperforms humans in terms of consistency, objectivity and speed.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
TIKKANEN, TUOMAS: Solujen tunnistaminen mikroskooppikuvista Histogram of Oriented Gradients menetelmällä
Diplomityö, 50 sivua
Marraskuu 2014
Pääaaine: Laskennallinen systeemibiologia
Tarkastajat: Yliopistonlehtori Heikki Huttunen & Tutkijatohtori Pekka Ruusuvuori
Avainsanat: Solujen tunnistaminen mikroskooppikuvista, Histogram of Oriented Gradients, Kasvukäyrä, Koneoppiminen, Tukivektorikone, Hahmontunnistus, Digitaalinen kuvankäsittely

Tässä diplomityössä tutkitaan solujen tunnistamista mikroskooppikuvista käyttäen Histogram of Oriented Gradients (HOG) piirteitä. Hahmontunnisprosessiin HOG piirteitä käyttäen sisältyy luokitus lineaarisella tukivektorikoneella (SVM), joka on ohjatun oppimisen luokitinmalli. Työssä esitetyssä menetelmässä luokitinmalli opetetaan iteratiivisesti etsien vaikeita esimerkkejä. Menetelmän tarkkuutta tarkastellaan käyttäen opetusvaiheessa 16:ta kuvaa ja testausvaiheessa 12:ta kuvaa. Mikroskooppikuvat on otettu kirkaskenttäoptiikalla (bright-field microscopy) ihmisen eturauhassyöpäsoluviljelmästä (PC3), mihin on merkitty puoliautomaattisesti yhteensä 10736 solua. Menetelmän tarkkuutta arvioidaan käyttäen käyttäen Receiver Operating Characteristic käyriä, sekä F_1-score ja Bivariate Similarity Index metriikoita. Päätös oikean ja väärän tunnistuksen välillä tehdään käyttäen PAS-metriikkaa.

Tulokset osoittavat, että HOG piirteitä voidaan käyttää onnistuneesti solujen tunnistuksessa mikroskooppikuvista. Peräkkäisinä päivinä soluviljelmästä otetuista kuvista voidaan esitetyn menetelmän avulla arvioida kasvukäyrä, jonka ennustamat solujen lukumäärät vastaavat suotuisasti käsin laskettujen solujen lukumääriä. Otettaessa vaikeat esimerkit mukaan opetukseen, väärien tunnistusten määrä pienenee huomattavasti. Vaikeiden esimerkkien etsinnän jälkeen luokittimen tarkkuus mitattuna ROC käyrän alle jäävänä pinta-alana saa korkeasta luokitustarkkuudesta kertovan arvon 0.98. Menetelmän toimivuuden puolesta puhuu myös F_1-score, jonka suuruus on 0.85 keskiarvoistettuna kaikkien testikuvien yli, kun kullekin kuvalle ideaalista SVM herkkyyttä on sovellettu. Työn johtopäätöksenä todetaan, että toteutettu menetelmä laskee solut riittävällä tarkkuudella sekä objektiivisemmin että nopeammin kuin ihmiset ja on näin ollen varteenotettava lähestymistapa automaattisessa solujen tunnistamisessa mikroskooppikuvista.

PREFACE

The master's thesis you are about to read - or most probably browse through - contains a report of the work that has been carried out at the Department of Signal Processing, Tampere University of Technology (TUT), during 2014. Regardless of the ups and downs along the way, I have found the research fascinating and enlightening. I consider this as magnificent development compared to my situation in 2009, when I started to study signal processing while having no idea what I was getting into.

I would say there were two important occasions during my university studies. First, realizing one could put computer science and cell biology together, resulting in computational systems biology, which addresses questions fundamental to our understanding of life. Even though it is not the easiest subject to study, it brings me a great deal of pleasure to think that I could be working with something actually useful. Secondly, it was not until very late in my studies that I fully understood and discovered the true potential in machine learning. That is why I am very pleased that I had a chance to prepare this master's thesis, which combines these two fields of studies.

I really appreciate all the people who helped me with this project. First of all, I would like to thank my supervisors Heikki Huttunen and Pekka Ruusuvuori who provided the research topic and many invaluable advices. Whenever I ran into problems, they had always some solution to overcome the difficulties. Additionally, I appreciate Heikki's role as University Lecturer for introducing me the field of signal processing. Secondly, I would like to thank Postdoctoral Researcher Leena Latonen who provided the microscope images and guidance related to cell biology matters. Latonen is a member of the Molecular Biology of Prostate Cancer group (MBPCG) which is led by Professor Tapio Visakorpi at the University of Tampere. Thirdly, I thank inspiring lecturer and former colleague Hanna Silén for giving me the possibility to work as a research assistant, which ultimately lead to this master's thesis.

Last but not least, I wish to express my gratitude to all the colleagues that I did not mention already, friends, family members, and my beloved girlfriend for support and thesis improvement suggestions. Finally, I thank coffee breaks and you, the reader. This thesis is dedicated to everyone who will find the study helpful.

14th of November 2014 in Tampere, Finland

CONTENTS

ABBREVIATIONS AND ACRONYMS

AUC	Area Under Curve
BSI	Bivariate Similarity Index
CV	Cross-Validation
DNA	Deoxyribonucleic acid
FN	False Negative
FP	False Positive
FPR	False Positive Rate
HOG	Histogram of Oriented Gradients
PAS	Localization metric used in PASCAL VOC
PC3	Prostate Cancer cell line
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TPR	True Positive Rate

1. INTRODUCTION

A cell is the basic unit of all known living organisms. Normal cells grow, divide to produce more cells, and die in a controlled way that is regulated by the body [1]. Cells contain DNA, which is a blueprint for your physical body. If a series of changes happen in the genes that are responsible for the regulation of the cell cycle, it can lead to cancer. Cancer is a group of diseases where cells start to grow and divide abnormally. Rate of cell division can be observed with a *growth curve*, which is a useful statistical tool in cancer treatment research. Growth curve is an empirical model, providing a way to study the evolution of cancer cell culture over time in terms of number of cells [2].

How can we acquire the number of cells in a given set of samples in the purpose of creating growth curve? In a case of a limited number of samples to be analyzed, one of the most straightforward *cell counting* method is to manually count the cells by eye under a microscope. However, manual cell counting is an extremely time-consuming process because it is difficult to process more than a portion of the sample by the eye. Humans are also subjective and we tend to get tired at the end of the day. Fortunately, people discovered that computers could be used to automatically count the number of cells in microscope images. To quantify the number of cells, they have to be somehow recognized first. This task is important and challenging image processing problem in bioimage informatics [3], which is referred to as *cell segmentation* or *cell detection*. Cell segmentation aims at identifying cell boundaries from multi-cell images, whereas in cell detection the object is to detect only the *location* instead of the *area* of each cell. Cell segmentation gives more detailed information about cells in an image, but from cell counting's point of view, there is no need for more than localization of cells provided by cell detection.

The first steps of automated cell counting were taken in 1966 [4]. The year before, Intel co-founder Gordon E. Moore, stated that the number of transistors in an integrated circuit would double approximately every two years [5]. The prediction has proven to be accurate and has had its impact also on medical imaging systems. This rapid development in technology has led to a situation where high-throughput imaging systems are generating such large quantities of data that computer-assisted analysis has become essentially required. Automated cell counting does not only ensure objectivity and consistency [6] but it is also efficient because computers can

process large volumes of data quickly and do not get tired or bored.

The purpose of this thesis is to study the suitability of cell detection from bright-field microscope images using Histogram of Oriented Gradients (HOG) feature descriptors in order to create growth curve. There are several reasons why HOG was chosen to address the problem. First of all, HOG has been successfully applied to a number of computer vision problems [7; 8; 9]. Secondly, HOG can represent partially occluded objects. Thirdly, the original paper pointed out its superiority in performance when compared to other popular object detection methods such as SIFT [10]. Finally, HOG has also been used as the basis of more advanced detection approaches such as the state-of-the-art Latent SVM object detection algorithm [11].

Research of a similar nature has been conducted before, which concludes that HOG can be successfully applied to the human cell detection [12]. This thesis validates the result and aims to give deeper insight into the topic by providing a practical evaluation, where the method is trained and applied to a research-scale large collection of human prostate cancer cell images spanning a six day cell culturing experiment.

Several methods exist for counting the cells in an image. The difficulty of performing the cell segmentation or detection task and choosing the correct algorithm depends much on the type of cells being targeted. If the cells are well separated from each other and have uniform intensity, simple thresholding or watershed algorithms are popular choices of approach [13; 14]. If the cells are packed together, algorithms which account for cell shape and size are preferred [15]. Cells are usually more packed together in tissue samples than those grown in culture. Another approach, like the one used in this study, is to recognize cells using training-based machine learning methods. Examples of cells and background are shown to the detector, which then learns their most important characteristics. In general, both supervised and unsupervised learning algorithms can be applied.

Microscope images usually vary more or less from application to application, for example, in terms of cell shape, cell density, noise level or resolution [16]. There are also many different microscopy techniques that are used with cell imaging. For example, in fluorescence microscopy, cells are first stained with a fluorescent dye and then illuminated with high energy light which causes the target cells to glow as their stains emit lower energy light. Even though fluorescence labeling can enable more accurate and effective automated analysis, it is not applicable in all use cases because fluorescence is not permanent and staining causes unnecessary stress to cells. These disadvantages can be overcome by using the simplest light microscopy technique, bright-field microscopy, combined with image processing methods. A bright-field microscope simply measures visible light transmitted through a sample.

Cell segmentation and cell detection are very application specific tasks and uni-

versal solution does not exist. The holy grail of software development in this field of study is to develop set of tools which could be used successfully in extracting cell-based data from different types of images.

The rest of this thesis is structured as follows. Chapter 2 gives an introduction to the field of machine learning and pattern recognition, it explains the structure of a typical pattern recognition system, and describes the model selection tools that are used in this study. Chapter 3 presents the theory of object detection with HOG features in detail. The metrics that are applied in performance evaluation are explained in Chapter 4. Chapter 5 discusses practical considerations in the implementation and introduces both the structure of the implemented cell detection framework and microscope images. Results are presented in Chapter 6, which addresses classification accuracy and overall performance of the system. Finally, Chapter 7 discusses meaning of the results and potential future work.

2. PATTERN RECOGNITION AND MACHINE LEARNING

Object recognition is easy for humans - even little children can do it. For example, when we are shown images of chairs, we immediately know it is a chair even though we have never seen a chair like that. It does not matter much if the image is fuzzy, if we see only half of the chair, or if the chair has four legs or only one leg. How do we know the image represents chair? We have seen a lot of different chairs and captured what is essential in them. When we were children we saw objects and someone told us some of them were chairs. Sometimes we had to be corrected when we called something chair that was actually not a chair. In the same way, if we want a computer to recognize chairs from images, we first have to teach them how chairs and not-chairs look like. Until this day, object recognition unfortunately remains as largely unsolved problem for computers.

After computers were invented, people started wondering if they could be programmed to learn like humans. That would mean automatically learning how to learn by themselves and improving with experience. We do not know how to do it yet. Machine learning as a branch of artificial intelligence is, however, working towards the goal by studying how we can teach computers to make and improve predictions or actions based on some data. The data could be, for example, thousands of images of human faces as the computer learns to recognize faces, or recordings of conversations in order to understand speech. The more the examples, the better the computer learns. Luckily we live in a world where enormous amounts of data are being collected every day. Thus, automated procedures are required when making sense of this diverse collection of data. Computers can already outperform humans in many logical problem solving tasks, such as playing chess. But in perceptual processes, computers perform a lot worse than humans, even though recent advances in specific learning tasks claim that computers can solve some tasks even better than humans [17]. It seems that machine learning will become increasingly important in computer science and part of mainstream technology.

When it comes to current trends in machine learning, neural networks or more specifically deep and large networks have attracted a great interest during recent years. Instead of conventionally concentrating on creating complex and sophisticated features, we can employ deep learning that can automatically learn the features if

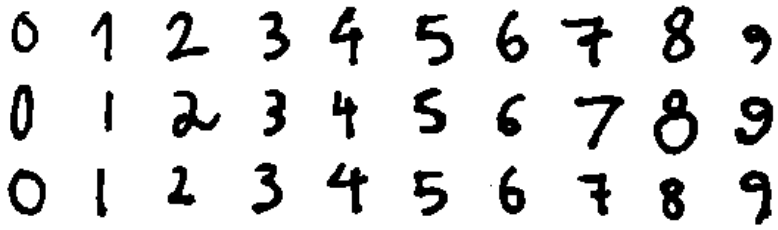

Figure 2.1: Examples of handwritten digits.

large amounts of training data is available. Using these computationally intensive techniques has resulted in a tremendous improvement in image recognition accuracy [18; 19].

Some people talk about machine learning as pattern recognition, where the goal is to teach computer to recognize patterns from the data. Example of pattern recognition task is using purchase transaction data to identify groups of customers who behave in a similar way. However, what is essential to these nearly synonymous branches, is their basis being in statistical learning theory. Statistical learning theory provides a framework for making optimal decisions given all the information available [20]. It is worth noting, though, that the information can be incomplete or ambiguous.

The basic building blocks of machine learning and pattern recognition systems can be introduced by considering an example problem of recognizing handwritten digits, such as the ones shown in Figure 2.1. Let's say we have collected a set of N images, where each represents one handwritten digit. Images are processed so that each image will be represented by a *feature vector* $\boldsymbol{x} = (f_1, f_2, \ldots, f_M)$ of M *features* (*e.g.*, the width of the letter in the image) along with *class label* y, which expresses the identity of the digit $(0, \ldots, 9)$. Set of feature vectors and class labels $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$ is referred to as *training data*, which is the input of a learning algorithm. The goal of the learning is to construct a rule, or *classifier*, which can be used to predict the most probable class label $\hat{y}_u$ of feature vector $\boldsymbol{x}_u$ representing unseen image with index u of handwritten digit. The ability to classify correctly unseen data, referred to as *testing data*, determines the level of *generalization*. The problem of generalization is a central issue in machine learning and pattern recognition.

The main types of learning systems are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, we have prior information about what kind of example belongs to what class. In unsupervised learning, we have to look for interesting patterns or clusters in the data without any feedback about the correctness of the findings. In other words, in supervised learning class labels are known while they are unknown in unsupervised

learning. Semi-supervised learning combines labeled and unlabeled data to achieve improvement in learning accuracy when compared to a situation where only labeled or unlabeled data would have been used. In reinforcement learning, a software agent interacts with its environment and receives a reward after every action. The agent keeps track of these rewards, and over time it will learn what action to choose in order to gain the largest reward.

The two most common supervised learning tasks are classification and regression. Classification predicts class labels whereas regression aims to find some function which describes the data so that real-valued numbers can be predicted. Regression is done by choosing a model and analyzing the relationship between its dependent variable and one or more independent variables. The most common unsupervised learning task is clustering, where the data is split into subgroups so that data points inside one subgroup are somehow more similar to each other than data points between different subgroups.

2.1 Basic Structure of Pattern Recognition Systems

There exists a variety of approaches in machine learning and pattern recognition. Similarly, there are a vast number of problems that can be solved with these systems. Regardless of the fact that the majority of different systems have very application specific pipelines, many of them can be thought to consist of five key stages [21]:

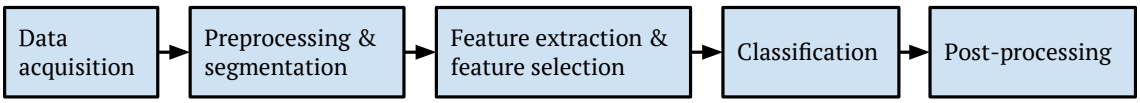

Figure 2.2: Basic structure of pattern recognition systems consists of five key stages: data acquisition, preprocessing and segmentation, feature extraction and feature selection, classification, and post-processing.

Data acquisition refers to collecting observations of the environment of interest somehow. Sensing can be done for example simply by counting the number of cars passing by every hour, or with any measurement device such as recording electrical activity of the brain with an EEG machine or taking images with a microscope, like the material was collected for this thesis. It usually happens that measurements capture also some noise or other irrelevant data that has nothing to do with the pattern recognition task. Computer vision is machine learning related field which focuses on images captured from the real world in order to produce numerical or symbolic information.

Preprocessing and segmentation reduce the variability between classes of interest. **Preprocessing** refers to operations performed on the raw data in order to improve its quality. Such operations could be filtering background noise or normalizing the

data. Normalization means rescaling the data to a standard scale, such as $[0, \ldots, 1]$, so that it becomes comparable. For instance, temperature can be acceptably measured in both Celsius and Fahrenheit, but those units are not comparable until normalized.

Segmentation refers to act of dividing data into smaller pieces of usually fixed size, so that each piece represents one object of interest. Each microscope image in this thesis contains multiple cancer cells. Each cell is therefore manually segmented into individual image before feature extraction.

The aim of **feature extraction** (FE) is to transform the data into some new *feature space*, where the pattern recognition problem becomes easier to solve. More specifically, FE is the search of such features, which best describe the classes in the data. Unfortunately, universally optimal features do not exist. FE is an application specific task; some features might work well with one problem, but poorly with another. For example, let's think of a task of distinguishing apples from oranges. They are both more or less round, so feature describing their shape would not help us much to distinguish them from each other. A lot better choice would be to select color as a feature. Output of FE is a feature vector containing the extracted features.

Could we use raw data as features? Nothing stops us from doing that. However, let's say our facial recognition system's material consists of 1000×1000 pixel grayscale images. If each pixel represented one feature, it would lead us having feature vectors of length 10^6, which may be computationally infeasible. Feature vectors would also assumingly contain many features useless for classification. FE aims to find a feasible number of features which will extract the relevant information from the input data. One of the classic methods is Principal Component Analysis (PCA). The goal of PCA is to select uncorrelated variables (principal components) from the data, which best describe the variation in a sum-squared error sense [21]. One has to be careful when reducing dimensionality in this stage. Accuracy of the system can suffer if the information important to the solution is discarded.

If there is a reason to suppose that the feature vector contains many redundant or irrelevant features, **feature selection** (FS) can be used to select only a subset of features while discarding the rest. Benefits of FS are easier interpretation of the predictive model, reduced computational effort in the training stage and better generalization due to reduced overfitting. Example of FS method is LASSO, which minimizes the residual sum of squares subject to the sum of the absolute value of the coefficients being less than a constant [22]. As a result, irrelevant features will be discarded as their coefficients become zero.

Feature extraction and feature selection are closely related but different processes, even though some consider them as one broad category of methods and sometimes FE and FS methods are included in classifier implementations. However, the objec-

tive of FE is to transform the existing features into a lower dimensional space by creating new features whereas FS selects a subset of the existing features.

In the **classification** stage, the goal is to determine the most appropriate class for given new feature vector based on training data. The algorithm which implements the classification is called the classifier. Variety of different classifiers exist. Some of the most popular ones such as neural networks, support vector machines, k-nearest neighbors, Bayesian classifiers, decision trees and logistic regression are explained in [20; 21] in detail. The range of algorithms raises a question: how do you know which classifier suits the best for the particular classification problem? Of course, some classifiers are computationally less complex than others, but it is usually more important to have adequately large, high-quality data set than selecting appropriate classifier. One approach is to perform cross-validation over a number of different classifiers and select the one which minimizes the classification error.

Classifiers can be divided into two categories, generative and discriminative. Generative techniques build a probabilistic model for the distribution of the features, whereas discriminative techniques directly learn a mapping between inputs and class labels [23]. Example of a generative model is Gaussian Mixture Model and example of a discriminative model is Support Vector Machine, which is used in this study. Both techniques have their own characteristics, strengths, and weaknesses. Generative models are often more flexible but it has been shown that discriminative classifiers can outperform them if large enough training data set is available [24].

Post-processing refers to the actions taken based on the result(s) of classification. In the case of character recognition this could mean forming words from single letters and performing spell check validation.

2.2 Model Selection

Model selection is a process of selecting appropriate machine learning techniques and parameter values for given data. Usual workflow starts with selecting a set of candidate models. Next, search strategy and an evaluation criterion are selected. The candidate models are compared by building the models according to the search strategy and calculating the evaluation criterion values for each model. Finally, the model is selected which performs the best. Subsection 2.2.1 discuss grid search as an example of search strategy and Subsection 2.2.2 discusses cross-validation as an example of evaluation criterion.

In the model selection task the complexity of the candidate models is tuned so that the models will generalize, capture the essence of the data, as well as possible. Complexity can mean, for example, describing the data with an equation with more parameters than actually needed. When the model is too complex, usually overfitting occurs. Overfitting is one of the central problems in machine learning.

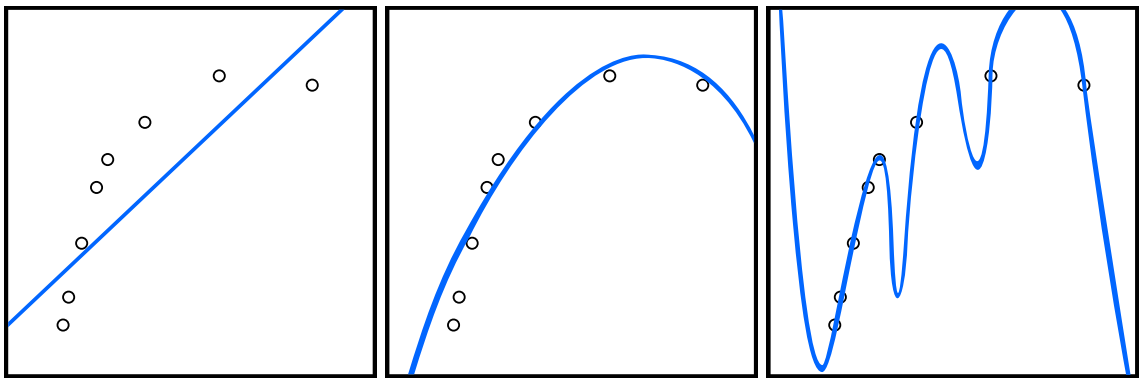

Figure 2.3: Example of model selection in regression. Left: the model is too simple and underfitting occurs. Center: well generalized model capturing the essence of the signal. Right: the model is too complex and overfitting occurs.

Overfitted model describes noise or random error instead of the underlying signal in the data. On the contrary, underfitting occurs when the model fails to capture the underlying signal in the data. Underfitted model is usually too simple. Both underfitted and overfitted models will lead to poor predictions on unseen data. Figure 2.3 illustrates underfitted model, well-generalized model and overfitted model.

Bias and variance are important statistical properties describing the quality of the model. Their nature is easy to explain with an example. Let's think of an archer shooting arrows at a target. Bias describes how much the archer systematically misses the bullseye in the same direction. Variance describes how scattered the arrows are. Underfitted model has typically high bias and low variance. Well-generalized model typically has low bias and low variance. Overfitted model has typically low bias and high variance.

A medieval monk, William of Occam, stated in the 14$^{\text{th}}$ century: "Entities should not be multiplied unnecessarily.". The principle became known as Occam's razor. It proposes that a problem should be stated in its basic and simplest terms. It can be applied to model selection meaning the simplest model that fits the data is also the most plausible [25].

2.2.1 Grid Search

Grid search is a method for parameter optimization. It starts by selecting reasonable and usually exponentially growing sequences (*e.g.*, $10^{-5}, 10^{-4}, \ldots, 10^{4}, 10^{5}$ or $2^{-10}, 2^{-9}, \ldots, 2^{9}, 2^{10}$) for each parameter. All combinations of parameter values are used one after another to build a new model. Performance metric values are calculated for each model. As a result, the combination of these parameter values with the best score is chosen.

The downside of the grid search is the computational time required to find suitable

values for a given parameter. One solution to speed up the grid search is to do a coarse search first in order to identify "better" region of the grid. Next, finer grid search is performed in this small region.

2.2.2 Cross-Validation

Cross-validation (CV) is a model evaluation method. It is essential to be able to ensure the quality of a model and to indicate future model predictivity on unseen data. There are many types of CV techniques, but all of them have the same basic idea. First, data is split into multiple parts. In Figure 2.4, which illustrates cross-validation, data is divided into 5 parts. Then, one part of the data (training set, parts 1,2,4 and 5 in Figure 2.4) is used for training a model and other part of data that was not used in training of the particular model (validation set, part 3 in Figure 2.4) is used for testing the model. Testing means measuring performance, *i.e.*, CV error. If the same data was used for training and testing, it would result in biased classification results. It would be the same thing as giving away the answers when arranging an exam. Using independent data set gives a hint of the level of generalization of the model and helps to avoid overfitting.

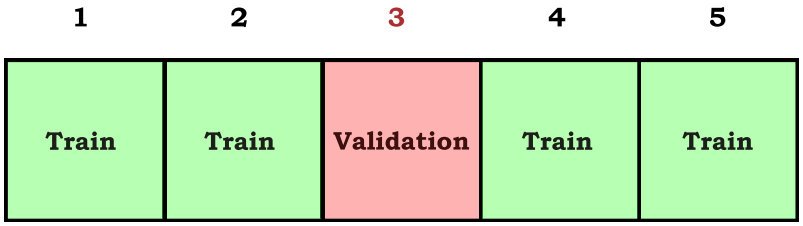

Figure 2.4: Illustration of cross-validation.

In k-fold cross-validation, the data is randomly split into k parts of equal size. One of the parts is reserved for testing and the rest k-1 parts will be used for training. The CV process is repeated k times (the folds) so that on every iteration different part is used for testing. The final model that is selected is the one which produces the best performance averaged over all k folds.

Leave-one-out cross-validation (LOO-CV) is the same as k-fold cross-validation with the exception that k=n, where n is the total number of examples. The benefits of LOO-CV are avoiding random sampling and making maximum use of the data. The disadvantage of LOO-CV is high computational cost because n different models are trained on all the data except for one example.

3. OBJECT DETECTION

The goal of object detection is to detect all instances of a particular class in given image or video sequence. The task is still a challenge in the field of computer vision mainly because of the number of possible sources of variation. There are a large number of possible locations where objects can appear at different scales and shapes. Even if two objects from the same class are more or less similar sized in real life, but have different distance from the camera, they appear at different scales in the image. Objects can be partially obstructed from view or objects from different classes can somehow remind each other too much, which leads to false detections. Additionally, variation in the imaging process such as changes in illumination, changes in camera position, or digital artifacts has to be taken into account.

This chapter is organized as follows. Section 3.1 describes step by step how objects are detected using Histogram of Oriented Gradients (HOG) feature descriptors. Section 3.2 explains how integral images can be used to speed up HOG feature computation. Section 3.3 introduces main ideas behind Support Vector Machine, which is the last step in object detection with HOG. HOG is regarded as a discriminative object detection method because SVM learns explicit boundary between classes.

3.1 Histogram of Oriented Gradients

Dalal and Triggs introduced Histogram of Oriented Gradients (HOG) feature descriptor in 2005 [7]. HOG describes features based on local histograms of gradient orientations weighted with gradient magnitudes. In the original paper, the method was proven to be effective in human detection, but HOG can also be used successfully with other pattern recognition problems such as face recognition or vehicle orientation detection [8; 9].

HOG has become very popular after its release in the field of computer vision. One of the key advantages of HOG is being able to describe object orientation while showing invariance to geometric and photometric transformations because it operates in localized regions. In other words, HOG tends to be unaffected by changes in shapes and lighting, which appear in larger spatial regions. HOG can be applied to both color and grayscale images, but using color gives slightly better results.

An overview of object detection with HOG is presented in Figure 3.1. HOG consists of gamma and color normalization, gradient and orientation computation,

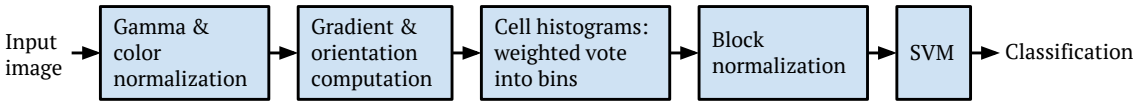

Figure 3.1: Block diagram of object detection with HOG feature descriptors.

cell histogram computation, normalization across blocks, and flattening into a feature vector. After calculating the HOG feature, it is classified with support vector machine. Following Subsections 3.1.1-3.1.5 explain these steps in detail.

3.1.1 Gamma and Color Normalization

The first step in HOG is to perform gamma and color normalization. These preprocessing procedures bring a modest improvement in performance and reduce the noise introduced by the camera. The noise in the image distorts gradient values which are computed in the next step. Color normalization, or intensity normalization when targeting grayscale image, changes the range of pixel intensity values to achieve consistency in dynamic range between images. Following equation defines the color normalization I_N [26]:

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}, \tag{3.1}$$

where I is the image to be normalized, Min and Max are its minimum and maximum values, and newMin and newMax define the range in the normalized image.

Gamma γ, in turn, represents nonlinear relationship $L_{actual} = L_{detected}^{\gamma}$ between detected light $L_{detected}$ (*i.e.*, pixel value) and actual luminance L_{actual} [27]. When a number of photons is doubled, also the signal received by digital camera sensor is doubled. However, human visual system behaves differently: when number of photons is doubled, we perceive the light as being only a fraction brighter. Simple gamma normalization is done by taking the square root of the pixel values in the image [7]. This operation expects the gamma to be 0.5. In grayscale images, gamma and color normalization are applied on intensity values and in color images they are applied to each color channel.

3.1.2 Gradient and Orientation Computation

The gradient tells how the image changes in the given direction. Gradient computation is done by sliding a mask over the image pixel by pixel in x and y directions while calculating gradient value for each pixel describing relationship of neighboring pixel values according to the mask. Dalal and Triggs compared the functioning of

following masks:

$$
\begin{array}{lll}
\textit{1-D centered:} & \nabla_x = [-1, 0, 1] & \nabla_y = [-1, 0, 1]^\mathsf{T} \\
\textit{1-D uncentered:} & \nabla_x = [-1, 1] & \nabla_y = [-1, 1]^\mathsf{T} \\
\textit{1-D cubic-corrected:} & \nabla_x = [1, -8, 0, 8, -1] & \nabla_y = [1, -8, 0, 8, -1]^\mathsf{T} \\
\textit{3x3 Sobel:} & \nabla_x = \begin{bmatrix} -1, & 0, & 1 \\ -2, & 0, & 2 \\ -1 & 0, & 1, \end{bmatrix} & \nabla_y = \begin{bmatrix} -1, & -2, & -1 \\ 0, & 0, & 0 \\ 1, & 2, & 1 \end{bmatrix} \\
\textit{Roberts cross:} & \nabla_x = \begin{bmatrix} 1, & 0 \\ 0, & -1 \end{bmatrix} & \nabla_y = \begin{bmatrix} 0, & 1 \\ -1, & 0 \end{bmatrix}.
\end{array}
\tag{3.2}
$$

The simple *1-D centered* mask worked the best with human detection. When using this mask, x-gradient's first and last column and y-gradient's first and last row have to be handled separately because the mask does not wholly fit in the image in those locations. Thus, the difference between the edge pixel and its adjacent pixel is calculated. The result of the gradient computation with the *1-D centered* mask is presented in Figure 3.2. Also, effect of Gaussian smoothing before applying the mask was studied, but it was noticed to lower performance.

Figure 3.2: There is the original image on the left, followed by x-gradient in the center and y-gradient on the right. Gray pixels have small gradient while black and white pixels represent a large gradient.

Image gradient has two properties: *magnitude* and *orientation*. The *magnitude* tells how quickly the image is changing and the *orientation* tells the direction in which the image is changing most rapidly. They are defined as follows:

$$
\textit{magnitude} = \sqrt{\nabla_x^2 + \nabla_y^2} \qquad \textit{orientation} = \arctan\left(\frac{\nabla_y}{\nabla_x}\right). \tag{3.3}
$$

In the case of grayscale image the gradient is calculated from the pixel intensity values, whereas in color images, gradient is calculated for each channel and the one with with the greatest magnitude (and its corresponding orientation) is selected.

3.1.3 Cell Histograms

Next, the image is divided into small subsections called cells. For each cell, a histogram of the gradient orientations of the pixels inside the cell is calculated. The orientations can be between 0-180° (unsigned) or 0-360° (signed). Unsigned orientations worked out better with human detection. Dimensionality is reduced by quantizing orientations into a predefined number of evenly spaced bins (discrete intervals). Dalal and Triggs noticed that increasing the number of bins increases performance up to 9 bins, after which increasing bins makes only little difference. Each pixel in the cell casts a vote weighted by its magnitude for the bin that its orientation was quantized to. The voting simply means increasing the frequency of the observed bin by the magnitude of the pixel.

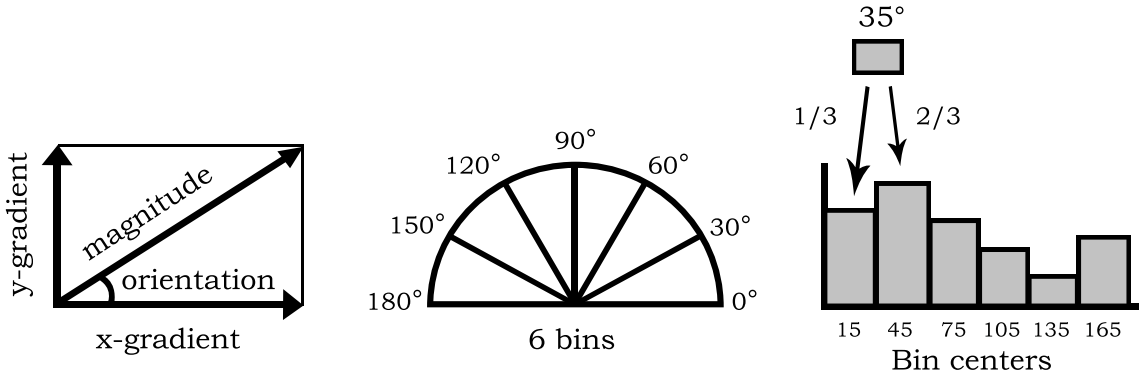

Figure 3.3: Left: magnitude and orientation are calculated for every pixel in the image from gradient values. Center: orientations are quantized to the bins. Right: weight is bilinearly interpolated between neighboring bins.

The accuracy of cell histograms is improved and aliasing is reduced by interpolating votes linearly between neighboring orientation bins and bilinearly into spatial cells. Linear interpolation between neighboring bins means dividing the vote between neighboring bins if the orientation does not happen to be exactly the same as one of the bin centers. For example, let's assume orientations between 0-180° are divided into 6 bins, and pixel gradient orientation of 35° is being added to the histogram. The difference is 10° to larger neighboring bin center and 20° to smaller neighboring bin center. Hence, $\frac{2}{3}$ of the magnitude is added to the larger neighboring bin and $\frac{1}{3}$ of the magnitude is added to the smaller neighboring bin. Figure 3.3 illustrates this example. Bilinear interpolation between spatial cells means distributing the vote between neighboring cell histograms according to the vertical and horizontal distance to those cell centers.

3.1.4 Block Normalization

After generating cell histograms, they are grouped together into larger subregions called blocks which typically overlap with each other. Figure 3.4 represents two different block geometries, which provide very similar performance: square or rectangular (R-HOG) and circular (C-HOG). R-HOG blocks are represented by three parameters: the number of cells per block, the number of pixels per cell and the number of channels per cell histogram. There are two C-HOG variants: one with a single central cell and another with angularly divided central cell. C-HOG blocks are described by four parameters: the number of angular and radial bins, the radius of the center bin and the expansion factor for the radius of additional radial bins.

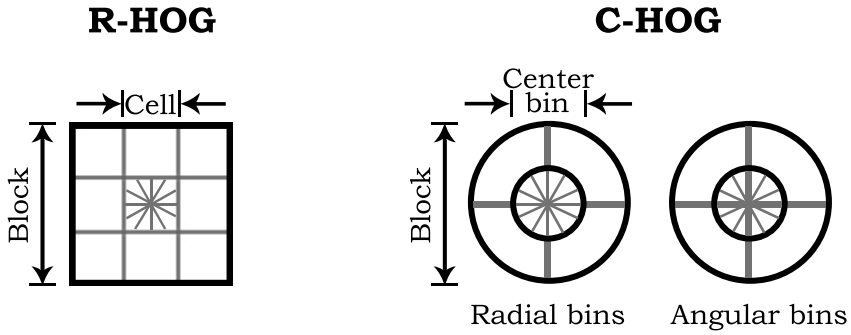

Figure 3.4: R-HOG and C-HOG block geometries.

The set of histograms inside each block is flattened to a 1-D feature vector v, which is then contrast normalized in order to increase performance by introducing better invariance to illumination and shadowing. Dalal and Triggs compared four different methods for block normalization: *L2-norm*, *L2-Hys*, *L1-sqrt*, and *L1-norm*. First three of these turned out to work equally well, whereas *L1-norm* produced a little less reliable performance. However, all four normalization methods improved performance significantly when compared to non-normalized data. *L1-norm*, *L1-sqrt*, and *L2-norm* are defined as follows:

$$\begin{aligned} &\textit{L1-norm of } v = \frac{v}{||v||_1 + \epsilon} \\ &\textit{L1-sqrt of } v = \sqrt{\frac{v}{||v||_1 + \epsilon}} \\ &\textit{L2-norm of } v = \frac{v}{\sqrt{||v||_2^2 + \epsilon^2}}, \end{aligned} \tag{3.4}$$

where $||v||_k$ is k-norm of v, and ϵ is arbitrary small number to prevent division by zero. *L2-Hys* is the same as *L2-norm* followed by clipping (limiting the maximum values of v to 0.2) and renormalizing, as in [10]. The final descriptor is obtained by concatenating the normalized block vectors to a single vector. The length of final

R-HOG descriptor is the number of cells in each block multiplied by the number of blocks in the image multiplied by the number of orientations, *i.e.*, quantized bins.

3.1.5 Classification with SVM

The last step in object detection with HOG is to use a supervised machine learning algorithm, Support Vector Machine (SVM). SVM is trained with HOG feature vectors of different classes. Hyperplane is adjusted accordingly to distinguish the kind of HOG features that each class typically exhibits. Linear kernel was proposed in the original paper, even though the use of Gaussian radial basis function kernel increases performance slightly the cost being increased run time.

When detecting objects from unseen image with a trained SVM classifier, sliding window technique is used. The input image is resized to various scales, through which optionally overlapping window is moved and descriptors are computed. For each window a score is assigned by the classifier, describing how probable the current detection is. As a result, multiple detections of the same object are achieved. These detections have to be fused into one detection, for example, with mean-shift method.

The resizing process gives the ability to detect objects at multiple scales with a single model. Computationally more expensive method would mean training a model for each scale and running them on the same image without resizing.

3.2 Integral Image

The computational effort of HOG can be reduced by precomputing integral images from gradient magnitudes for each bin [28]. It allows cell histograms to be built in constant time, O(1) time complexity, by extracting sums of rectangular subregions from the integral image. Integral image is also known as Summed area table, which was introduced to computer graphics already in 1984. However, the algorithm was not properly introduced in computer vision until 2001 by Paul Viola and Michael Jones [29]. Integral image *ii(x,y)* describes sums of all original image pixel-values *i(x',y')* left of and above each pixel *(x, y)*:

$$ii(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y'). \tag{3.5}$$

Figure 3.5 shows an example where integral images are generated from gradient magnitude values for each bin. Gradient orientations 0-180° are quantized to 6 bins. It can be seen how integral images get brighter towards the lower right corner at different rates depending on orientation bin indicating the accumulation of magnitude values.

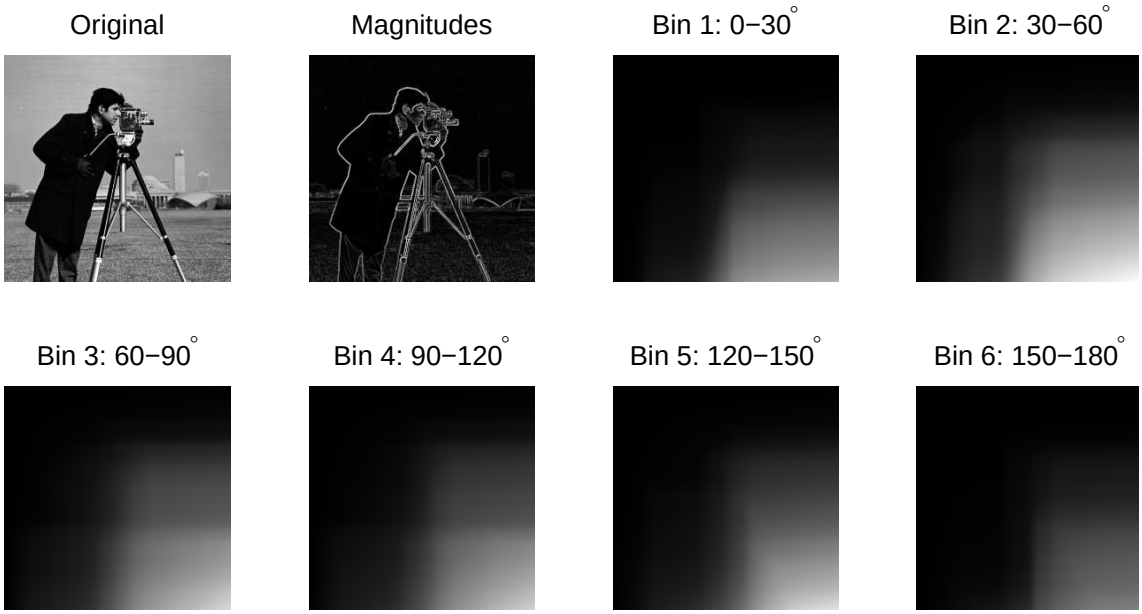

Figure 3.5: The first two leftmost images in the upper row are the original image and its gradient magnitudes. Subsequent images are integral images calculated from gradient magnitudes for each bin. Intensities of integral images are normalized to the same scale.

After computing the integral image, sums of arbitrary sized rectangular subregions of original image pixel-values can be calculated using only four array references into the integral image:

$$\sum_{\substack{x_0<x\leq x_1 \\ y_0<y\leq y_1}} i(x,y) = ii(\mathrm{L}_4) + ii(\mathrm{L}_1) - ii(\mathrm{L}_2) - ii(\mathrm{L}_3), \tag{3.6}$$

where L_1=(x_0, y_0), L_2=(x_1, y_0), L_3=(x_0, y_1) and L_4=(x_1, y_1). Figure 3.6 illustrates the computation of rectangle D spanned by the points L_1, L_2, L_3, and L_4.

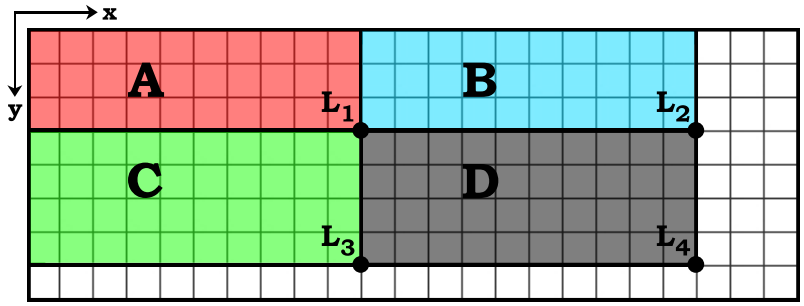

Figure 3.6: The sum within rectangle D in the original image can be computed with the help of four locations L_1, L_2, L_3 and L_4 in the integral image. Locations and the sum of pixels in rectangles: L_1=A, L_2=A+B, L_3=A+C, and L_4=A+B+C+D. The sum within D can be computed as L_4+L_1-L_2-L_3.

3.3 Support Vector Machine

Corinna Cortes and Vladimir Vapnik introduced supervised machine learning algorithm Support vector machine (SVM) in 1995 [30]. SVM is highly robust and computationally effective classifier which provide a learning technique for pattern recognition and regression estimation. The idea behind SVM is to maximize the margin around the separating hyperplane between two classes. Until this day, SVM remains as one of the most popular classification methods in machine learning [31].

This section is divided into three subsections. Subsection 3.3.1 introduces the optimization problem of finding the maximum margin hyperplane. Subsection 3.3.2 describes kernel functions, which transform the data into higher dimensional space where non-separable data becomes separable. Lastly, subsection 3.3.3 discusses practices in preprocessing and parameter selection when implementing SVM classifier.

3.3.1 The Optimization Problem

Let's assume Figure 3.7 like classification problem of separating two classes with linear hyperplane H. We are given n training examples $\{\boldsymbol{x_i}, y_i\}, i = 1, \ldots, l$, where each example has d features ($\boldsymbol{x_i} \in \mathbb{R}^d$) and a class label ($y_i \in \{-1, 1\}$). The hyperplane separates the data points with $y_i = -1$ from those with $y_i = 1$. Other methods might find lots of possible solutions, but SVM finds an optimal solution. It maximizes the distance γ, or margin, between the hyperplane and the nearest data points from both classes (the support vectors). All hyperplanes are defined as the set of points $\boldsymbol{x}$ which satisfy

$$H: \quad \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b = 0, \tag{3.7}$$

where $\boldsymbol{w}$ is the vector perpendicular to the hyperplane and bias b is a constant representing the distance from the origin to the hyperplane along the direction of $\boldsymbol{w}$. If the data is linearly separable, it is possible to form mutually parallel hyperplanes H_1 and H_2 so that they separate the data without leaving any data points between them:

$$\begin{cases} H_1: & \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b = -1 \\ H_2: & \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b = +1. \end{cases} \tag{3.8}$$

Data points on the H_1 and H_2 are the support vectors and the space between H_1 and H_2 is the margin. Following two constraints ensure that arbitrary input $\boldsymbol{x_i}$ will not fall into the margin:

$$\begin{cases} \boldsymbol{w}^\mathsf{T}\boldsymbol{x_i} + b \leq -1 & \text{for } y_i = -1 \\ \boldsymbol{w}^\mathsf{T}\boldsymbol{x_i} + b \geq +1 & \text{for } y_i = +1, \end{cases} \tag{3.9}$$

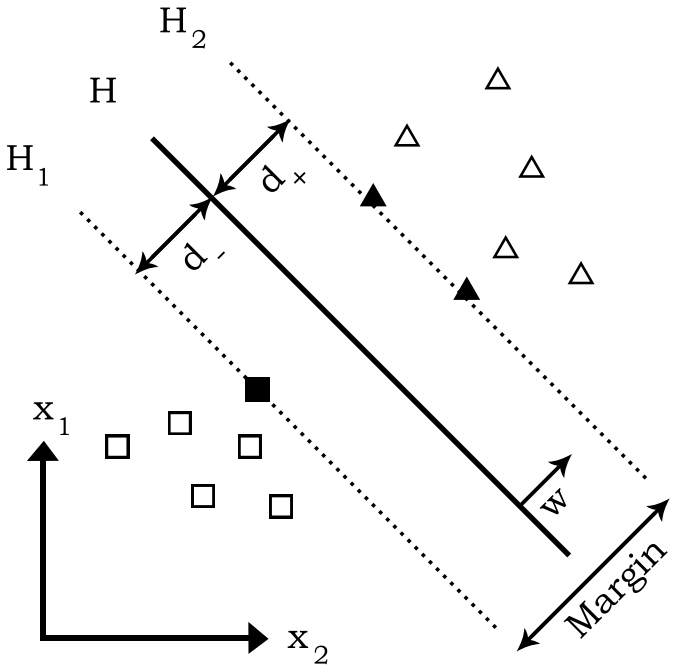

Figure 3.7: Linear and optimal hyperplane H separates class squares from class triangles with maximum-margin. Margin is maximized with the help of parallel hyperplanes H_1 and H_2 pressing against support vectors, closest samples of each class, that are marked with solid black.

which can be combined together as:

$$y_i(\boldsymbol{w}^\mathsf{T}\boldsymbol{x_i} + b) - 1 \geq 0 \quad \forall i. \tag{3.10}$$

Let d_- be the shortest distance from the separating hyperplane to the closest negative examples and d_+ the shortest distance from the separating hyperplane to the closest positive examples. Now, we can define the margin γ, or distance between H_1 and H_2, as $d_- + d_+$.

The points which lie on the hyperplane H_1 have perpendicular distance $\frac{|-1-b|}{||\boldsymbol{w}||}$ from the origin, where $||\boldsymbol{w}|| = \sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{w}} = \sqrt{\boldsymbol{w_1^2} + \boldsymbol{w_2^2}}$. Similarly, the points which lie on the hyperplane H_2 have perpendicular distance $\frac{|1-b|}{||\boldsymbol{w}||}$ from the origin [32]. Because the separating hyperplane is equally distant from H_1 and H_2, it will give us $d_- = d_+ = \frac{1}{||w||}$. Furthermore, this gives the margin

$$\gamma = d_- + d_+ = \frac{2}{||\boldsymbol{w}||}. \tag{3.11}$$

In order to maximize the margin, optimal parameters $\boldsymbol{w}$ and b are found by minimizing $||\boldsymbol{w}||^2$ subject to the constraints in equation 3.9. This is a constrained optimization problem, which is solvable by Lagrangian multiplier method. The dual

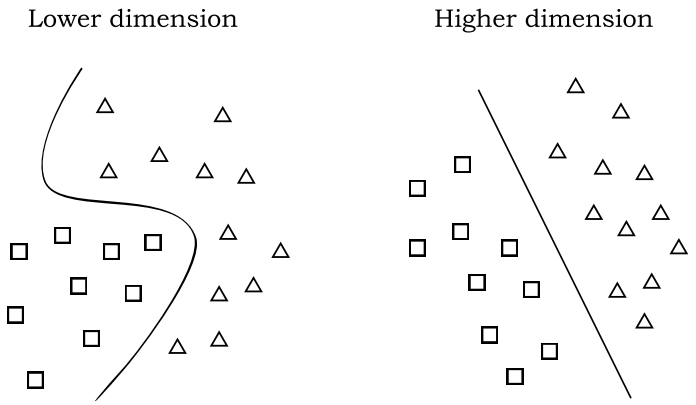

Figure 3.8: The kernel function maps lower dimensional input space to higher dimensional feature space where the data becomes linearly separable.

form of the C-SVM problem is:

$$\min \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) - C \sum_{i=1}^{l} \alpha_i, \tag{3.12}$$

subject to $\alpha_i \geq 0$ and $\boldsymbol{y}^\mathsf{T} \alpha = 0$ where α_i represent Lagrange multipliers, $\boldsymbol{y} = [y_1, \ldots, y_l]^\mathsf{T}$, K is a kernel function, and C is cost parameter.

The main two formulations of SVM are C-SVM and ν-SVM. Even though they are different problems, they both have the same optimal solution set [33]. The formulations differ in terms of a penalty parameter. As their names imply, $C \in [0, \infty)$ is the penalty parameter in C-SVM, and $\nu \in [0, 1]$ is the penalty parameter in ν-SVM. The penalty parameters relax the constraints of the problem setting so that a solution can be found to non-separable data by allowing training errors. The parameter C controls the trade-off between training error and margin maximization. High value of C tries to classify all the samples correctly and the model may not generalize well to unseen data [32]. C can be seen as a way of controlling overfitting of the data. The parameter ν, in turn, represents an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors [34]. Some prefer ν-SVM over C-SVM because it has a more meaningful interpretation. C-SVM formulation is used in this thesis.

3.3.2 Kernel Functions

Sometimes linear classifiers are not complex enough to capture the structure in the data. In such case training examples $\boldsymbol{x_i}$ can be mapped with function $\phi : \mathbb{R}^n \to \mathbb{R}^m$ from n-dimensional space into higher m-dimensional space where it is possible to perform the separation. Figure 3.8 represents this mapping.

Kernel function K computes the dot product $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \phi(\boldsymbol{x_i})^\mathsf{T} \phi(\boldsymbol{x_j})$ in the

higher dimensional space without the need of more complex computation of the coordinates of the data in that space. This operation is called the kernel trick. Four basic kernels are:

$$\begin{aligned}
&\textit{Linear:} && K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i}^{\mathsf{T}} \boldsymbol{x_j} \\
&\textit{Polynomial:} && K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\gamma \boldsymbol{x_i}^{\mathsf{T}} \boldsymbol{x_j} + r)^d, \gamma > 0 \\
&\textit{Radial Basis Function (RBF):} && K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma ||\boldsymbol{x_i} - \boldsymbol{x_j}||^2), \gamma > 0 \\
&\textit{Sigmoid:} && K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(\gamma \boldsymbol{x_i}^{\mathsf{T}} \boldsymbol{x_j} + r),
\end{aligned} \tag{3.13}$$

where γ, r, and d are kernel parameters [35].

3.3.3 Data Preprocessing and Parameter Selection

SVM takes vectors of real numbers as an input. Therefore, if the input data has categorical features, such as {rock, paper, scissors}, it has to be converted into numeric data: {1,0,0}, {0,1,0}, and {0,0,1}. Additionally, it is highly recommended to normalize the range of the data (both training and testing data) linearly, for example, between $[-1, 1]$ or $[0, 1]$. Normalization prevents features in greater numeric ranges dominating those in smaller numeric ranges.

Choosing kernel function, its parameters, and parameter C is a crucial step in SVM when considering its efficiency. Generally the *linear* kernel provides a useful baseline, but in more sophisticated case cross-validation procedure is used to choose the best performing kernel function. Also, the RBF kernel is commonly suggested as a reasonable first choice because with certain parameters, it behaves like *linear* and *sigmoid* kernel [35]. Other reasons for choosing RBF are its suitable number of tunable parameters in terms of complexity of the optimization, and RBF kernel has fewer numerical difficulties than other kernels. However, RBF is not suitable if there are a vast number of features. When tuning RBF kernel, grid search of the parameters γ and C combined with cross-validation is a popular approach.

4. ACCURACY ASSESSMENT CRITERIA

In order to get an idea about how well the cancer cells are detected, it is necessary to study both SVM classifier performance and performance of the system as a whole. This is necessary because even if SVM is separating the cells from the background seemingly well, the detection process can produce much worse results than expected in case of badly chosen parameters for the sliding window procedure.

In this thesis Receiver Operating Characteristic (ROC) curves were used in assessing SVM performance. The system performance was measured with three other metrics. In the first approach, F-score was used. ROC and F-score are explained in Section 4.1. In the second approach, the number and locations of detected cells were compared with the actual number and locations of cells. To decide whether detections were correct or not, PAS localization evaluation metric was applied. The PAS metric is an acronym of PASCAL Visual Object Classes Challenge [36], from where it is adopted. The PAS metric is presented in Section 4.2. In the third approach performance was evaluated as a similarity between annotated binary ground truth image and its binary estimate image with the help of Bivariate Similarity Index, which is explained in Section 4.3.

4.1 Receiver Operating Characteristic & F-score

The cell detection is considered as a binary classification problem, where each instance I is mapped to either positive (cancer cell) or negative (something else than cancer cell) class label. Let's let labels $\{\boldsymbol{p}, \boldsymbol{n}\}$ represent the actual classes and labels $\{\boldsymbol{p'}, \boldsymbol{n'}\}$ represent the class predictions produced by a model. There are four possible outcomes when instance is classified. If the predicted label and the actual label are both positive, the instance is counted as *true positive* (TP). If the predicted label and the actual label are both negative, the instance is counted as *true negative* (TN). If the predicted label is positive and the actual label is negative, the instance is counted as *false positive* (FP). If the predicted label is negative and the actual label is positive, the instance is counted as *false negative* (FN). Table 4.1 presents the mentioned four different classification outcomes in a confusion matrix, which forms the basis for many common metrics. It is called confusion matrix because numbers along its diagonal from lower left to upper right present the confusion, or error, between classes. Well-performing classifier has most of the counts on the

Table 4.1: Confusion matrix presenting the possible outcomes of an individual classification.

		Actual class	
		p	***n***
Predicted class	***p′***	True positive	False positive
	n′	False negative	True negative
	Total	***P***	***N***

diagonal from upper left to lower right (correct classifications).

One of the simplest ways of measuring classification performance is *accuracy* - the fraction of correctly classified instances. It can be calculated from the confusion matrix as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \quad 0 \leq accuracy \leq 1. \tag{4.1}$$

Even though high *accuracy* is always a good thing, the number can be rather misleading. Let's consider an example of screening for a rare disease. One can be very accurate by blindly calling every case negative. If only 5 % of the patients have the disease, this kind of predicting method would lead to *accuracy* of 95 %. Because of the limited usefulness of *accuracy*, ROC and F-score are used here instead. They also take false alarms into account and thus provide better understanding of the performance.

Receiver operating characteristic (ROC) curve is a simple yet useful tool for analyzing detector performance. It helps in choosing the best model out of the set of candidate models. Put differently, ROC helps in deciding where to draw the line between the number of true positives (benefits) and false positives (costs). ROC curve plots false positive rate FPR

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}, \quad 0 \leq FPR \leq 1 \tag{4.2}$$

on x-axis against true positive rate TPR

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}, \quad 0 \leq TPR \leq 1 \tag{4.3}$$

on y-axis, while decision threshold or some other detector parameter is varied over a range of values [37]. In the case of creating ROC curve for given SVM model, varying threshold means varying the bias b term of the model. FPR is the fraction of FPs out of the total actual negatives and TPR is the fraction of TPs out of the total actual positives. TPR is also known as *sensitivity* or *recall* in machine learning.

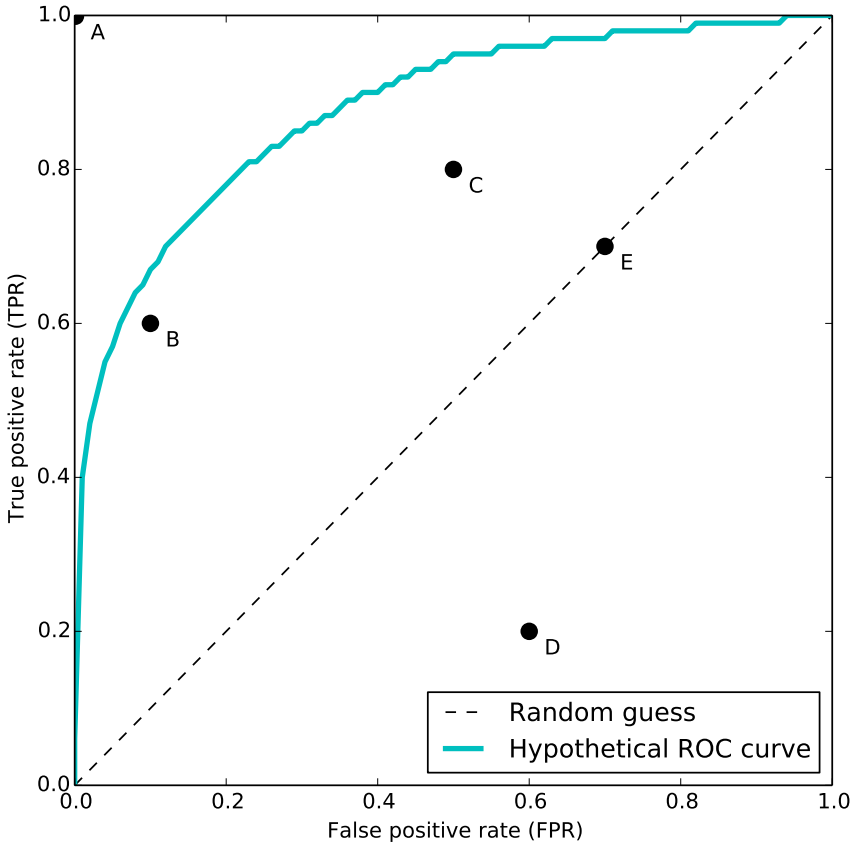

Figure 4.1: The ROC space with hypothetical ROC curve and scatter plot of the five prediction examples.

ROC curve was invented during World War II by British electrical and radar engineers. It was used as a tool to assess radar receiver operator's ability to distinguish whether a blip on the radar screen represented a flock of birds, friendly aircraft or enemy aircraft. Later, it has been employed in many different fields of studies such as signal detection theory, psychology, and machine learning [38].

An example shown in Figure 4.1 illustrates the ROC space with hypothetical ROC curve and scatter plot of five discrete classifiers labeled A through E. There are some points in ROC space which are important to notice. Classifiers located on the diagonal from (0,0) to (1,1), such as E, are worthless because they make as good predictions as made with a random guess. With random guess you are expected to get half the positives and half the negatives correct. Points above this diagonal represent better than random guess and points below the diagonal, such as D, represent worse than a random guess. However, in the case of binary classification, classification decision produced by classifier located below the random guess line can be reversed in order to utilize the power of the classifier, thereby producing a classifier located in the upper left triangle. Classifier A in Figure 4.1 at (0,1) represent a perfect classification with no false negatives no false positives.

Lower left corner of ROC space at (0,0) represent a situation where every instance is classified as negative. In such case there will be no false positives and no true positives. Its opposite is classifier located in upper right corner at (1,1) where every instance is classified as positive.

Classifiers located at the left-hand side of the ROC space are considered as "conservative". Such classifiers require strong evidence for positive classifications and thus produce only a small number of false positives. Classifiers located at the right-hand side of the ROC space are considered as "liberal". This kind of classifiers require less evidence for positive classifications and thus produce also more false positives than conservative classifiers. [37] Classifier B in Figure 4.1 is more conservative than classifier C. Many real-world problems have a large number of negative instances making conservative classifiers more attractive than liberal ones.

ROC curve can be reduced into a single scalar by calculating the area under the ROC curve (AUC). AUC enables performance comparison between different classifiers. It is defined in a following way:

$$AUC = \int_0^1 ROC(t)\ \mathrm{d}t, \tag{4.4}$$

where $ROC(t)$ is TPR and t is FPR. AUC can have values in range $[0, 1]$. AUC of a classifier is interpreted as the probability of ranking randomly chosen positive instance higher than randomly chosen negative instance. It is worth noting that even if an arbitrary classifier Y has lower AUC than arbitrary classifier Z, the classifier Y may still have better performance in some specific region of ROC space.

F-score measures performance using using *precision* and *recall* (TPR). *Precision* is defined as the fraction of the TPs out of the all the instances predicted as positive:

$$precision = \frac{TP}{TP + FP}, \quad 0 \leq precision \leq 1. \tag{4.5}$$

Precision can be interpreted as the probability of random positive prediction being correct, and *recall* can be interpreted as the probability of random positive instance being predicted as positive. In other words, *precision* tells more about the accuracy of the system, whereas *recall* tells more about the robustness of the system. F-score combines these two metrics into a single number. Its general definition is:

$$F_\beta = \frac{(\beta^2 + 1) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}, \tag{4.6}$$

where $0 \leq F_\beta \leq 1$ and $0 \leq \beta \leq +\infty$ [39]. β controls the relative importance of *recall* over *precision*. If *recall* is half as important as *precision*, β=0,5 is used. If *recall* is twice as important as *precision*, β=2 is used. The traditional way is to

give equal weights to both, leading to F_1-score:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}, \tag{4.7}$$

which calculates the harmonic mean of *precision* and *recall*. F_1-score values can also vary between $[0, 1]$. F_1-score can only be large when both *precision* and *recall* are large.

4.2 Definition of a Match

As mentioned in the previous section, each positively predicted label (cancer cell detection) can be counted either as TP or FP. In this thesis, the decision between TP and FP is based on studying the amount of spatial overlap $PAS(R_d, R_{gt})$ between rectangle detections R_d and rectangle ground truth cancer cells R_{gt}:

$$PAS(R_d, R_{gt}) = \frac{area(R_d \cap R_{gt})}{area(R_d \cup R_{gt})}, \tag{4.8}$$

where $area(R_d \cap R_{gt})$ corresponds to the number of pixels in the intersection of R_d and R_{gt}, and $area(R_d \cup R_{gt})$ corresponds to the number of pixels in the union of R_d and R_{gt} [36]. Values of $PAS(R_d, R_{gt})$ can vary between $[0, 1]$, where 1 corresponds to perfect localization. Detection is considered as TP only if its PAS metric exceeds arbitrary threshold value. In the original paper threshold value of 0.5 was used.

4.3 Bivariate Similarity Index

Paul Jaccard developed a method for comparing the similarity and diversity of ecological species in 1901 [40; 41]. The Jaccard index, or Jaccard similarity coefficient, $S_{J(T,E)}$ is defined as the size of the intersection of the reference data set T and its estimates E, divided by the size of the union of those:

$$S_{J(T,E)} = \frac{|T \cap E|}{|T \cup E|}, \tag{4.9}$$

where $0 \leq S_{J(T,E)} \leq 1$. If every data point is classified correctly, then $|T \cap E| = |T \cup E|$ and $S_{J(T,E)} = 1$. If the algorithm does not detect any cells, then $E = 0$ and $S_{J(T,E)} = 0$. The Jaccard distance, on the other hand, measures dissimilarity of two things and can be simply obtained from the Jaccard index as $D_{J(T,E)} = 1 - S_{J(T,E)}$.

The Jaccard index, however, can give a little biased view of the performance since it cannot distinguish certain underestimation cases from overestimation cases. Thus, an alternative pair of similarity measures, Bivariate Similarity Index (BSI),

have been proposed in [42]:

$$TET = \frac{|T \cap E|}{|T|}$$
$$TEE = \frac{|T \cap E|}{|E|}, \tag{4.10}$$

where $0 \leq TET \leq 1$ and $0 \leq TEE \leq 1$. If the estimate E is the same as reference T, then $TET = 1$ and $TEE = 1$. These similarity metrics divide performance in four sections:

- T and E dislocated from each other: TET and TEE are both small
- Overestimation: TET is large and TEE is small
- Underestimation: TET is small and TEE is large
- Good segmentation: TET and TEE are both large.

Sometimes it is more useful to have univariate metric instead of these bivariate indices. Therefore, segmentation distance d_{seg} is defined as the Euclidean distance from the point corresponding to the TET and TEE values to perfect segmentation where $TET = 1$ and $TEE = 1$:

$$d_{seg} = \sqrt{(1 - TET)^2 + (1 - TEE)^2}. \tag{4.11}$$

Even though d_{seg} does not provide information about overestimation or underestimation, it works as a general measure of segmentation accuracy.

5. THE CELL DETECTION FRAMEWORK

This chapter introduces the way, how the theory described in Chapters 3 and 4 is applied in practice in order to implement the cell detection framework. The framework is illustrated by the block diagram in Figure 5.1. It consists of separate training and testing phases. In the training phase, cell and non-cell (*i.e.*, background) example images are cropped from the input images using ground truth annotations. Then, HOG features are generated from each of the cropped images and an initial linear SVM classifier is trained, which is employed to iteratively learn final classifier by finding hard examples from the input images with sliding window method. In the testing phase, unseen images are scanned with sliding window and HOG features are generated from the position of each window in the image. These features are classified as cell or non-cell with the linear SVM classifier that was constructed in the training phase. It is likely that multiple positive detections will be clustered, because (a) sliding window procedure can be done in multiple scales, and (b) windows can overlap with each other on each scale. Thus, clustered detections are merged together. Finally, growth curve is created using an estimated number of cells in the images of each day.

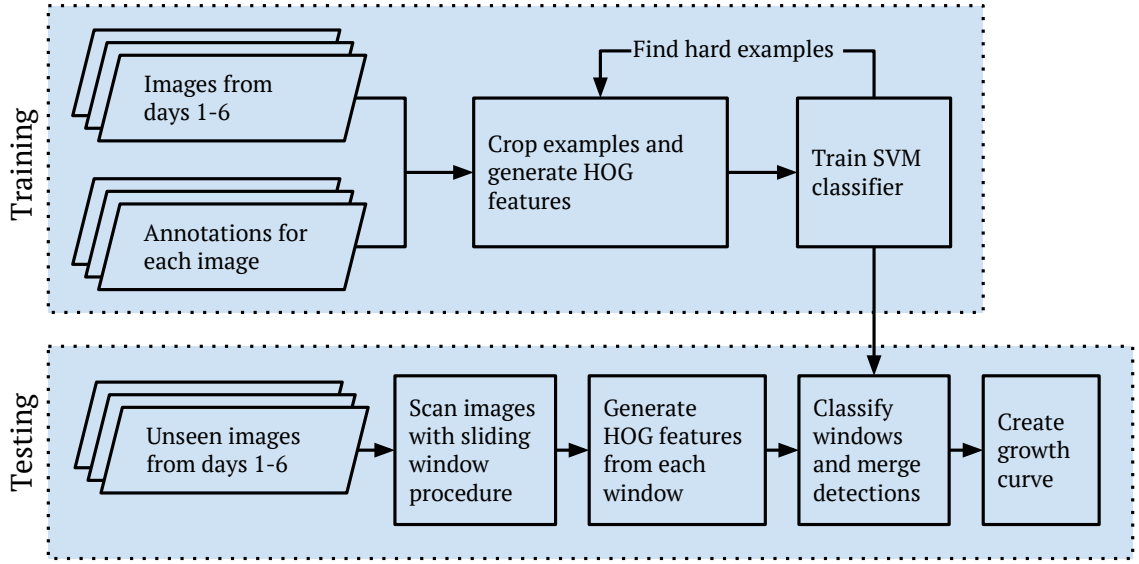

Figure 5.1: Block diagram of the implemented cell detection framework.

The outline of this chapter is as follows. Section 5.1 introduces the microscope images at which the cell detection framework was applied. Section 5.2 explains

how training data was collected and Section 5.3 explains how confusion matrix was created. Section 5.4 describes the choices made regarding software development tools.

5.1 Microscope Images

The cell detection framework was evaluated on a data set consisting of grayscale bright-field images of PC3 human prostate cancer cell lines. The PC3 cell lines were isolated from a bone metastasis of a 62-year-old Caucasian male in 1979. Images were taken with QImaging Retiga-2000R camera using Olympus IX71 microscope and Objective Imaging Surveyor automated scanning and imaging software. The cancer cells were kept in an incubator at constant 37 °C in 5 % CO_2 atmosphere while maintained in Ham's F12 nutrient mixture that was supplemented with 10 % FBS and L-glutamine. The cells were passaged using trypsin two times in a week to create more growing space. The day when the cells were passaged into the imaging chamber is called day 0. By the next day, the cells had attached to the bottom of the chamber and the sample was imaged. That day is called day 1. The data set consists of images taken on days 1-6. 192 images were taken with autofocus each day. Pixel resolution of the images is 1596 (width) × 1196 (height). Figure 5.2 presents cropped images of the same area in a sample throughout different days demonstrating the growth of cancer cell culture. It is worth noting that the cells could move around freely in the samples to some extent.

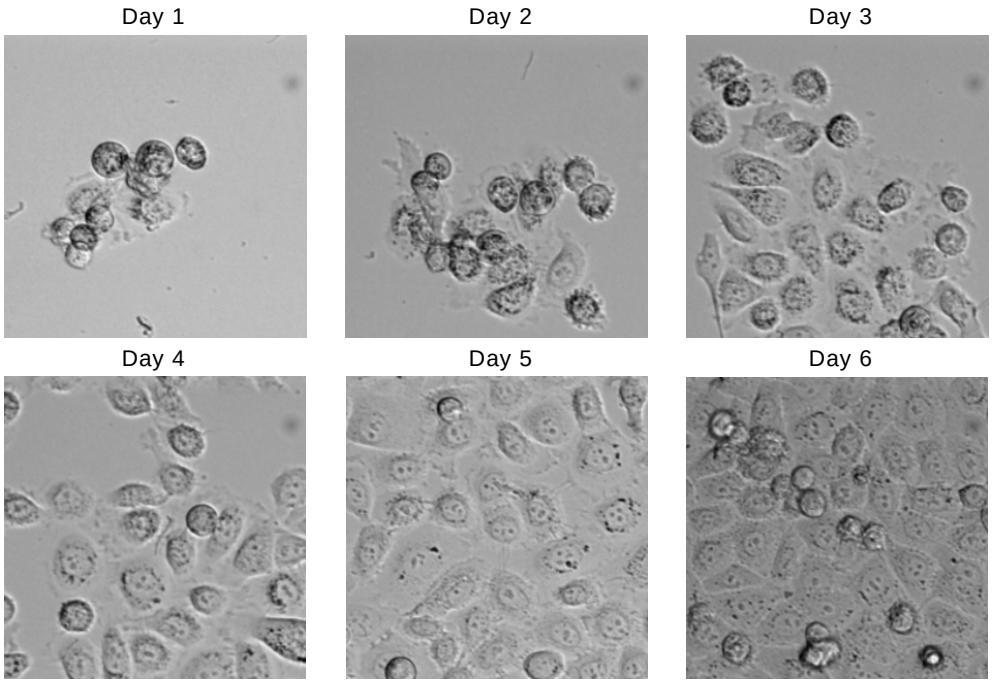

Figure 5.2: The data set consists of bright-field microscope images of PC3 cancer cell lines taken on days 1-6.

5.2 Training Phase

In order to achieve correct results, enough training samples must be collected. Here, the act of collecting training examples is referred as *annotating*, which was done by accurately separating representative rectangular areas from images. The objective was to collect training examples of each class from images of each day. Because manual segmentation is a tedious process, data was collected in a semi-automatic manner. First, smaller amount of training examples was collected manually to train the initial classifier, which was then employed to detect cancer cells from unseen images in order to collect larger amount of training examples to train a classifier for the first SVM training iteration. The initial amount of training data was collected by programming annotation software, with the help of which 840 individual cells (positive examples) in total were manually annotated from 6 day 1 images. Because every cancer cell in those images was annotated, it was possible to automatically collect the same amount of negative examples from random locations in *background*. In this case, background means every pixel in the image that is not part of any of the positive examples. For each annotated image, an *INI* file was created consisting of sections describing annotation coordinates and the class they represented. Figure 5.3 shows a screenshot of the annotation software on the left-hand side with one positive and one negative example. On the right-hand side, there is screenshot of INI file containing the coordinates and class identifiers for the rectangle annotations.

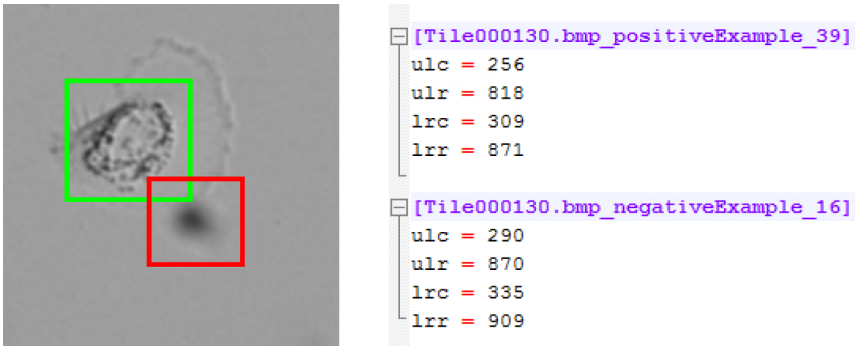

Figure 5.3: Left: cropped screenshot from the annotation software, including positive (green) and negative (red) annotation. Right: upper left corner (ulc & ulr) and lower right corner (lrc & lrr) coordinates of annotations are stored in sections in the INI file. Class labels of annotations are stored in section names.

Sizes of annotations can vary freely, but when HOG features are computed, training example images have to have the same size in order to produce equally sized feature vectors which SVM takes as input. That is why following steps are taken when cropping training examples from input images. If the aspect ratio of annotation is already the same as the aspect ratio of HOG descriptor window size, the

annotation is simply resized to match the window size. If the aspect ratio of annotation is different than the aspect ratio of the window size, smaller dimension (width or height) of the annotation is extended on both sides to correspond to the larger dimension. This procedure enables preservation of original aspect ratio. However, if the annotation is so close to the edge of the image that its dimensions cannot be extended, the annotation is cropped as such and resized to correspond to window size using bilinear interpolation. Now the original aspect ratio is lost and the image will be stretched, which does not necessarily help detecting similar cells.

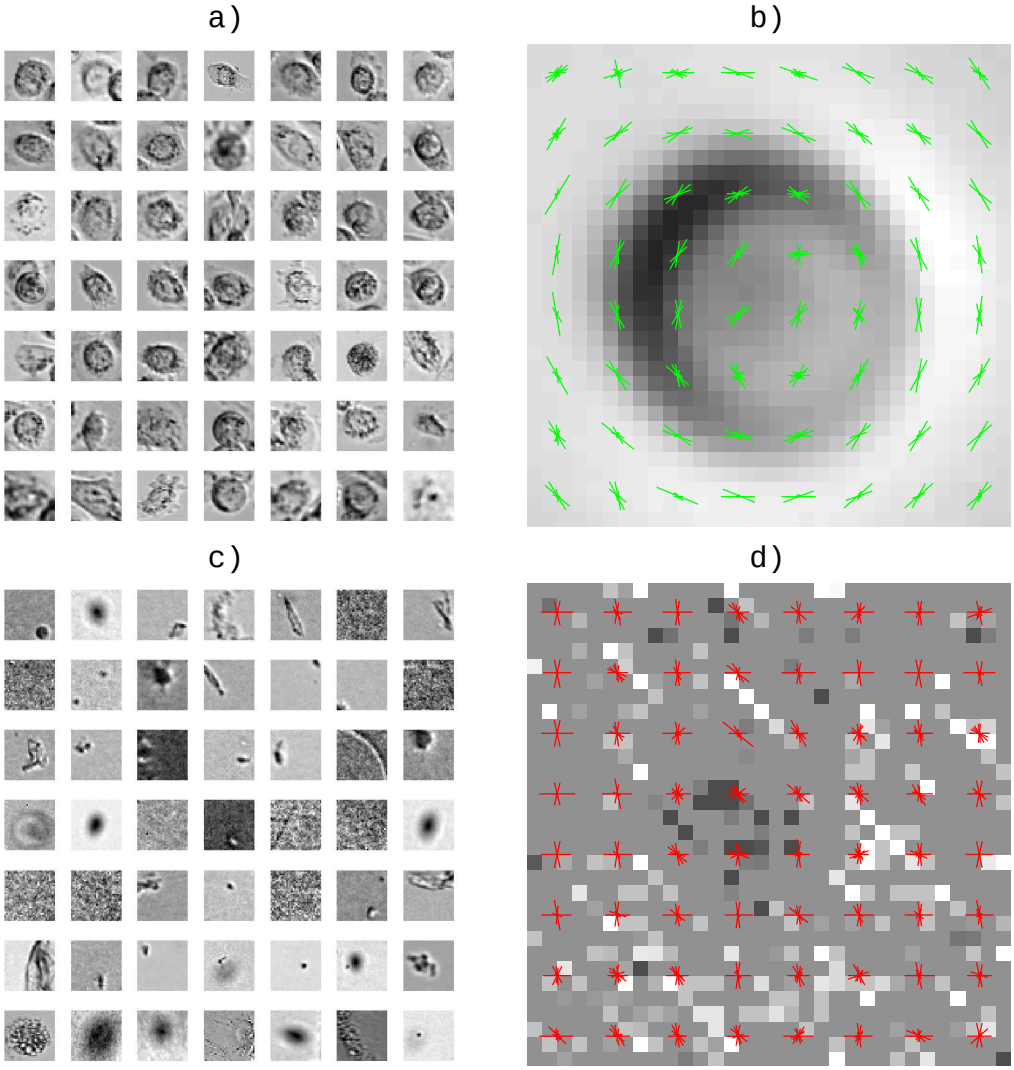

Figure 5.4: Illustration of experimental data: 49 randomly selected cell (a) and non-cell (c) example images, median image of all cell (b) and non-cell (d) examples and visualization of their HOG features. Images in a) and c) have been resized to 32×32 pixels and their intensities have been normalized to the same scale. Images b) and d) are also of size 32×32 pixels, but zoomed in versions of them are shown. Because pixel intensity values of d) varied within a narrow range when compared to that of b), pixel intensities of b) and d) have been normalized to different scales to provide better visualization.

Cell detection framework with the initial classifier was applied on images from days 2-6 (2 per day). The results were inspected with the annotation software to discard false positive detections. Also, more positive and negative examples were annotated manually. Like previously done with day 1 images, more negative examples were automatically collected from the background of day 2 and day 3 images. Automatic collection concerned only images from those days because not all cancer cells were annotated in subsequent day images. In other words, automatic collection from day 4-6 images would have caused some cancer cells to accidentally end up being marked as negative examples.

As a result, the total number of 4858 cell and 7198 non-cell examples were collected from 16 images from days 1-6 for the first SVM training iteration. Sizes of annotated PC3 cancer cells, written as mean $\pm$ standard deviation, vary in the images within a fairly narrow range: width 39 $\pm$ 8 px, height 38 $\pm$ 9 px, aspect ratio 0.9 $\pm$ 0.4 px. Each image represents an area of 1190.8 μm (width) x 891.4 μm (height) in real life. This magnification information gives roughly the actual sizes of cells: width 29.10 $\pm$ 5.97 μm and height 28.32 $\pm$ 6.71 μm. It is important to notice though, that the annotations were slightly bigger than the cells.

Figure 5.4 presents 32 $\times$ 32 pixel sized training examples from each class and median images of all training examples of each class with a visualization of HOG features. HOG features are visualized with rose plots, where each shows the distribution of gradient orientations within HOG cell. HOG features are visualized using cell size of 4 $\times$ 4 pixels. Rose plots consist of petals, which indicate the contribution of each orientation within the cell histogram. The median cell image consists of almost perfectly round object in the middle of the image, whereas the median non-cell image does not show any pattern and consists of low amplitude noise only.

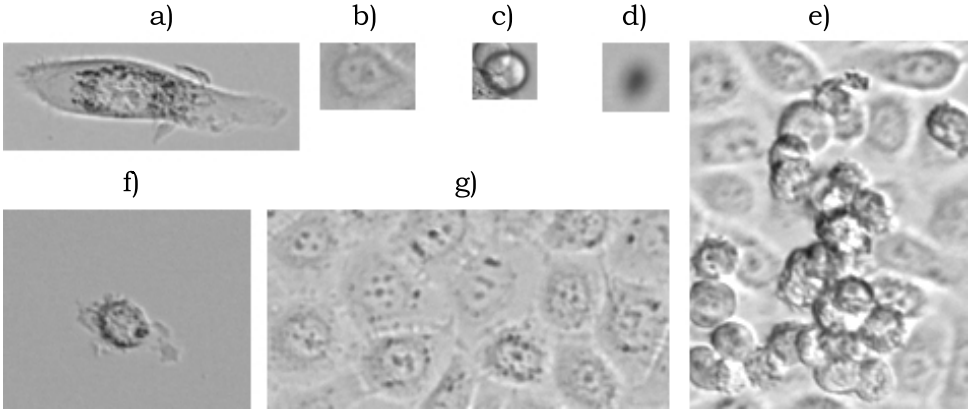

Figure 5.5: Examples of sources of variation in the microscope images. a) Large cell in focus, b) Average sized cell out of focus, c) Air bubble, d) Blurred spot caused by an unclean camera lens, e) Cells partially on top of other cells, f) Spatially isolated cell, g) Cells next to each other but not on top of other cells.

Annotating was not, and is not, straightforward process without expert knowledge in cell biology. There were multiple objects which were difficult to manually classify or segment because of different sources of variation in the microscope images. Examples of such sources of variation are presented in Figure 5.5. First of all, cells varied in terms of shape, size, and density. The easiest cells to segment were those that were spatially isolated from other cells. It was more difficult to segment cells that were right next to each other, and cells that were on top of other cells. The stacking of cells was mostly visible in day 6 images. It was caused by lack of horizontal and vertical space in the culture and indicated by stacked cells being on different focus level than those that were not stacked. Because of the high density of cells, it was inevitable that some of the positively labeled training examples included also parts of neighboring cells. When it comes to annotating partially occluded cells, only the cells which were approximately at least half visible were annotated. Other sources of variation complicating the training data collection were air bubbles in the samples on top of cells, cells out of focus, and blurred spots that appeared in the same locations throughout images. Blurred spots were caused by unclean camera lens. Additionally, it is possible that a few of the training examples were incorrectly labeled. It is likely that mentioned issues in the training data collection had a negative influence on the classification performance causing detection of false positives.

Detection of false positives was tried to overcome already in the training phase by training an initial classifier with all the training data from days 1-6 and using it to search for false positives, *i.e.*, "hard examples". All false detections that do not overlap with annotated cancer cells are considered as hard examples. This procedure is repeated multiple times and on each iteration hard examples are added to the training data. Hard examples are searched until the amount of hard examples is small enough. In the Chapter 6 iteration continues until less than 5 % of the initial amount of false positives are found. During the iterations, total of 2564 hard examples was collected for the final classifier. The hard examples that are found in the last iteration consist mostly of blurred spots that were caused by an unclean camera lens, shown in Figure 5.5 with label d). Finding hard examples was noticed to improve the classification performance considerably.

5.3 Testing Phase

A total of 5878 cancer cells were annotated manually from 12 images (2 per each day) for the purpose of testing the accuracy of the implemented cell detection framework. Confusion matrix is created by calculating PAS metric with each neighboring detection and ground truth cell. Two rectangles are considered as neighbors, if the distance between their horizontal and vertical center points do not exceed 200 pixels.

Those detections are counted as TP which have the PAS metric value larger than threshold 0.3. Figure 5.6 demonstrates the reason why threshold value 0.3 is used instead of 0.4 or 0.5, which was used in the PASCAL VOC challenge. Threshold values 0.5 and 0.4 require very exact localization, which causes many of the actually TP detections to be mistakenly counted as FPs. It is made sure that each annotated cell can have at most one matching true positive detection. The detections which do not have any matching annotations according to the mentioned constraints, are marked as FPs. The unmatched annotations represent the FNs. All pixels in the image that are not counted as TP, FP or FN represents TNs.

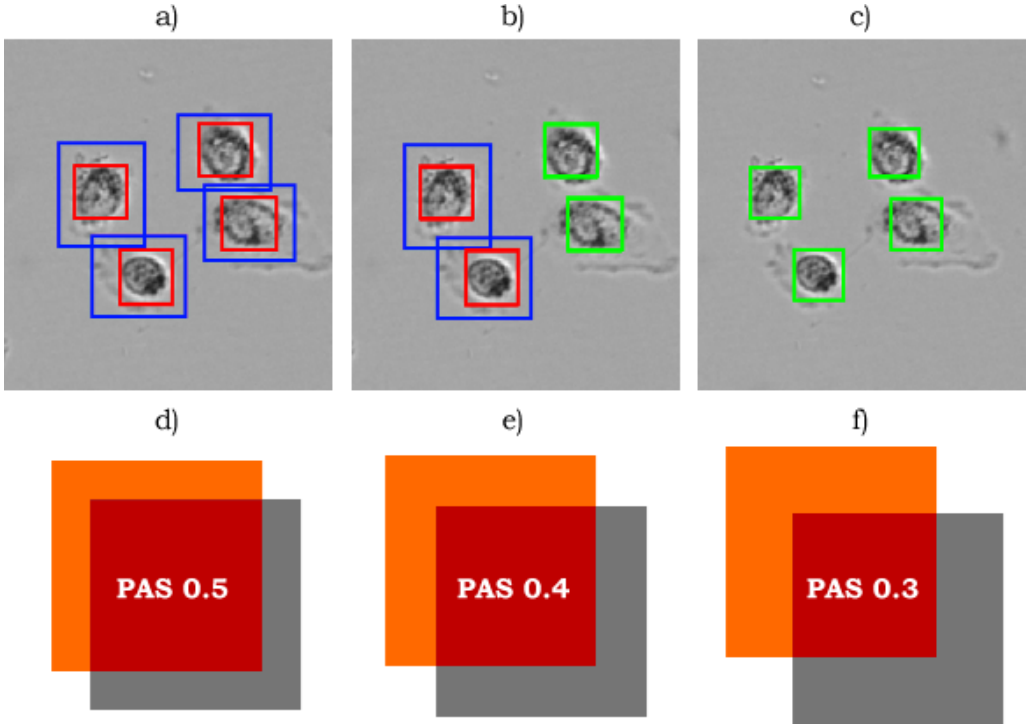

Figure 5.6: a), b), c): The same cropped area of the same sample with different (a: 0.5, b: 0.4, c: 0.3) PAS metric threshold values. Blue rectangles represent FNs, red rectangles represent FPs, and green rectangles represent TPs. With threshold value 0.5, all four detections are counted as FPs even though the detections are inside rather large annotations. With threshold value 0.4, two cells are considered as TP but the other two remain as FPs. With threshold value 0.3, all four cells are correctly counted as TP. Images d), e), and f) illustrate the amount of overlap (red) between orange and gray squares when PAS metric is 0.5, 0.4 and 0.3.

Detection of false positives is tried to overcome in testing phase by filtering initial detection results according to two constraints, which were assigned by subjectively inspecting the most common types of false positives. Detections completely inside other detections and detections overlapping with other detections more than the PAS metric value 0.8 are filtered out.

5.4 Software Implementation

The cell detection framework was implemented using *Python* (release 2.7.5) programming language. A Python wrapper for free *OpenCV* (release 2.4.9) library was used mainly to provide `HOGDescriptor` class, which implements HOG object detector [43]. OpenCV stands for Open Source Computer Vision. It was officially launched in 1999 by Intel Russia research center. OpenCV focuses primarily on real-time image processing, which requires rapid calculations. This is why it is written in optimized C++ code.

Table 5.1: OpenCV `HOGDescriptor` class and `detectMultiScale` function input and output parameters, their default values and descriptions.

		Parameter	Default value	Description
HOGDescriptor	input	winSize	(64, 128)	Descriptor window size
		blockSize	(16, 16)	R-HOG block size
		blockStride	(8, 8)	Block step size. It must be a multiple of cell size.
		cellSize	(8, 8)	Cell size
		nbins	9	Number of orientation bins
		derivAperture	1	The size of extended Sobel operator. Values & mask sizes: 1=1x3 & 3x1, 3=3x3, 5=5x5, 7=7x7.
		winSigma	-1.0	Gaussian smoothing window parameter
		histogramNormType	0	Normalization method, 0=L2-Hys
		L2HysThreshold	0.2	L2-Hys normalization method shrinkage
		gammaCorrection	True	Flag to specify whether the gamma correction preprocessing is required or not
		nlevels	64	Maximum number of detection window increases
	output	hog		HOGDescriptor class instance
detectMultiScale	input	img		Source image
		hitThreshold	0	Threshold for the distance between features and SVM classifying plane
		winStride	cellSize	Sliding/detection window step size. It must be a multiple of blockStride.
		padding	0	Extra padding (border) for image to detect objects partially outside image
		scale	1.05	Coefficient of the detection window increase
		finalThreshold	2	Grouping parameter for neighboring detections. 0 means not to perform grouping.
		useMeanshiftGrouping	False	Flag to specify whether to group detections using mean shift or “window overlap” method
	output	foundLocations		Coordinates of detected objects
		foundWeights		Weights (confidence) of detected objects

`HOGDescriptor`'s `detectMultiScale` function performs the sliding window procedure and merges detections. Table 5.1 presents descriptions of `HOGDescriptor` and `detectMultiScale` input and output parameters. The large number of the

parameters pose a challenge when studying the most suitable combination of their values in terms of the performance of the cell detection framework.

Based on the average dimensions of annotated cells that are presented in Section 5.2, winSize parameter of `HOGDescriptor` was fixed to (32,32) and padding parameter of `detectMultiScale` was fixed to (16, 16). Width and height of descriptor window size are power of two, which ensures computational efficiency (because computers are based on the binary numbering system, where the base is 2). Also, `detectMultiScale` only scales up the size of the detector window, meaning there cannot be smaller detections than descriptor window size. Padding was chosen to correspond to approximately half of the median size of positive annotations. That value was chosen because it enables the detection of cells, which are partially outside image.

Other important libraries in this work were *NumPy* (release 1.8) and *scikit-learn* (release 0.15.0b1) [44; 45]. NumPy provided multi-dimensional data structures and numerical operations on them. Scikit-learn is an open source machine learning library, which provided implementation of SVM, along with functions for cross-validation and ROC metric. The SVM in scikit-learn is implemented by wrapper around *LIBSVM* library [46].

To the best of our knowledge, there are no publicly available implementations of the complete SVM training procedure with proper mining for hard examples as described in the original HOG paper [7]. The implementation of this thesis along with the used data set can be downloaded from the supplementary website[1].

[1] `http://code.google.com/p/hog-cell-detection-framework/`

6. RESULTS

This chapter presents the results of the numerous experiments that were performed on different parts of the implemented cell detection framework. The experiments consisted of parameter testing and algorithm testing.

The rest of this chapter is separated into two main sections. Section 6.1 reports the results of studying SVM classification performance in the training phase of the cell detection framework and Section 6.2 reports the results of studying the overall performance in the testing phase of the framework.

6.1 Classification Performance

This section describes how well SVM functions in the proposed cell detection framework. It is important no notice that when SVM performance is evaluated in the training phase, input images are accurately segmented images of cells and background (referred as "easy examples"), which makes the classification task a lot easier than in testing phase. In testing phase the problem setting is more difficult because the input images can consist of, for example, half of a cancer cell or a quarter of it, depending on the location of sliding window in the image.

This section is structured as follows. Subsection 6.1.1 describes how the amount of training data affects SVM performance and Subsection 6.1.2 presents results of studying how a selected subset of `HOGDescriptor` parameters affect SVM performance. Subsection 6.1.3 presents how iterative training process affects the accuracy of SVM.

The cost parameter of each SVM classifier presented under Subsections 6.1.1 and 6.1.2 have been cross-validated on the range of $10^{-4}, 10^{-3}, \ldots, 10^{3}, 10^{4}$ so that on each fold training set consisted of all the images except the ones taken on a specific day (LOO-CV). Also, in the middle of each ROC graph under mentioned subsections, there is a zoomed-in image from the upper left corner of the ROC space. On the right-hand side of each ROC graph, there is a table showing the `HOGDescriptor` parameter values which remained the same when performing the particular test. The length of feature vector is denoted by "fvl" in the legend of the ROC graphs.

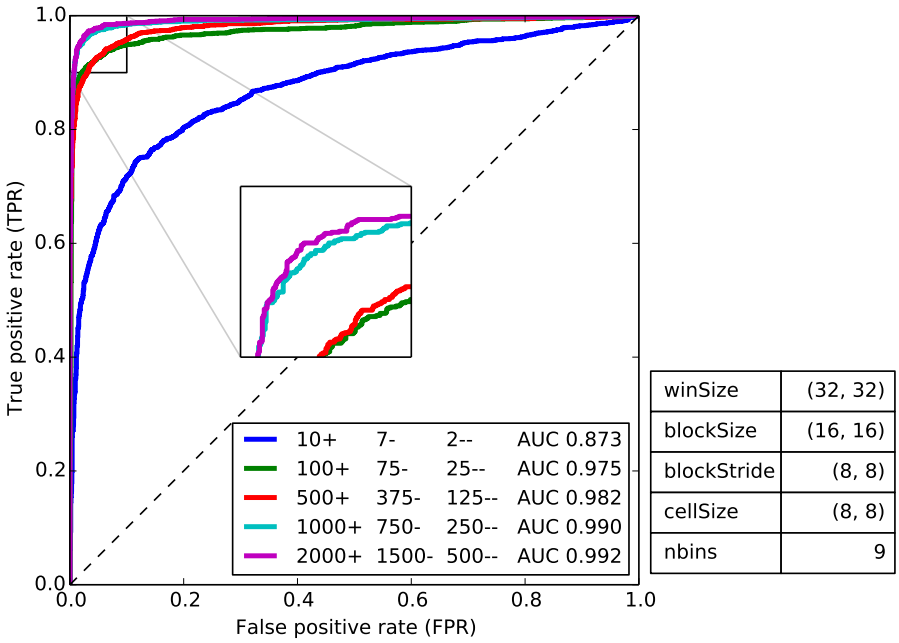

Figure 6.1: ROC curves for different numbers of training examples. "+" denotes positive examples, "−" denotes negative examples, and "−−" denotes hard negative examples. ROC curves were calculated by classifying separate 2000 positive and 2000 negative examples with no hard examples.

6.1.1 Effect of Amount Of Training Data

ROC curves in Figure 6.1 show that when the number of training examples is increased, also the SVM performance is improved. Using only 10 positive examples and 10 negative examples is already yielding moderate classification result with AUC being 0.873. AUC is increased by ∼0.1 to 0.975 when the amount of training data increased by tenfold. When 500 positive examples and 500 negative examples are used in training, AUC reaches the value of 0.982. Excellent prediction with AUC of 0.990 is produced when 1000 positive examples and 1000 negative examples are used in training. Finally, when 2000 positives and 2000 negatives are utilized, SVM performance increases with a lower rate than before to AUC of 0.992.

The Figure 6.1 implies that when more than 10 training examples from each class are used, TPR grows rapidly to at least 0.9 while FPR stays less than 0.1 when the threshold is lowered from the situation where every instance is classified as negative. However, the growth of TPR levels off after its rapid increase, meaning the cost of increasing TPR after this point is the rapid growth of FPR.

6.1.2 Effect of HOG Parameters

This subsection presents the results of determining the most suitable combination of `HOGDescriptor` parameter values. Following parameters are studied in the same

order as they are listed here: blockSize, blockStride, cellSize and nbins. Every ROC graph under this subsection has been created by using separate training and testing sets, which both consisted of 2000 positive and 2000 negative examples with no hard examples included.

ROC curves in Figure 6.2 show how SVM accuracy is affected when block size is varied. The smallest and the largest studied block sizes (4,4) and (32,32) produce equally sized and shortest feature vectors with the length of 576. Those block sizes result also in worst AUC scores of 0.989 and 0.988. SVM performs the second best with AUC of 0.994 when the block size is (16,16). The best AUC of 0.997 is obtained with block size (8,8), which has width and height corresponding to a quarter of those of the window size. It is worth noting that SVM performs worse with the block size (16,16) than with the block size (8,8) even though twice as long feature vector (length 3600) is produced when the block size (16,16) is used.

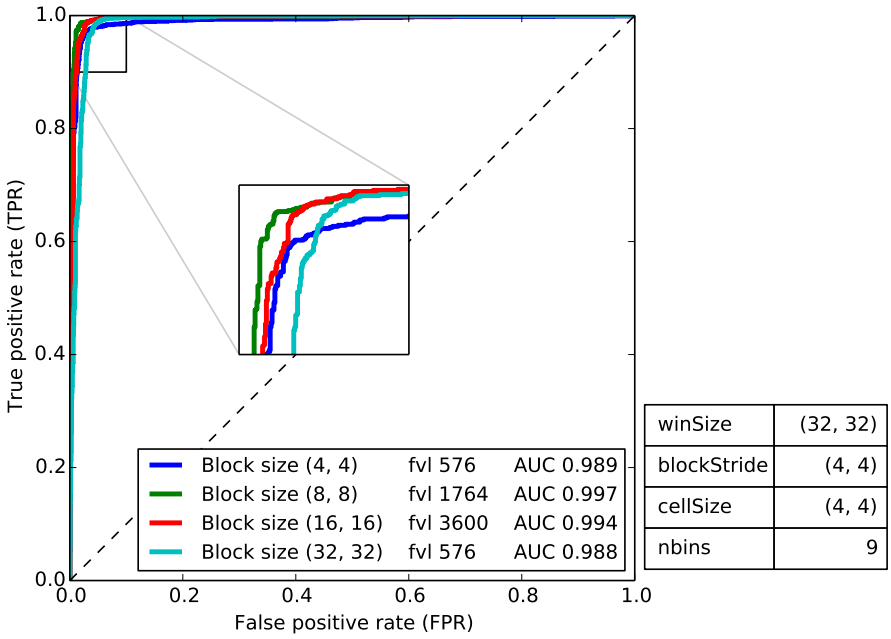

Figure 6.2: ROC curves for different block sizes.

When it comes to choosing the most suitable block stride parameter value, ROC curves in Figure 6.3 imply that too small or too large block stride should not be selected. The best AUC of 0.996 is obtained when the block stride (4,4) is used. The width and height of the block stride (4,4) correspond to half of those of the used block size. Block strides (2,2) and (8,8) yield 0.002 lower AUC of 0.994. Block stride (8,8) equals to having no overlap between adjacent blocks because the block size was attached to (8,8) when the results were generated.

ROC curves in Figure 6.4 show that SVM performance is improved when smaller cell size is used. The best AUC of 0.995 is obtained when cell size (2,2) is used.

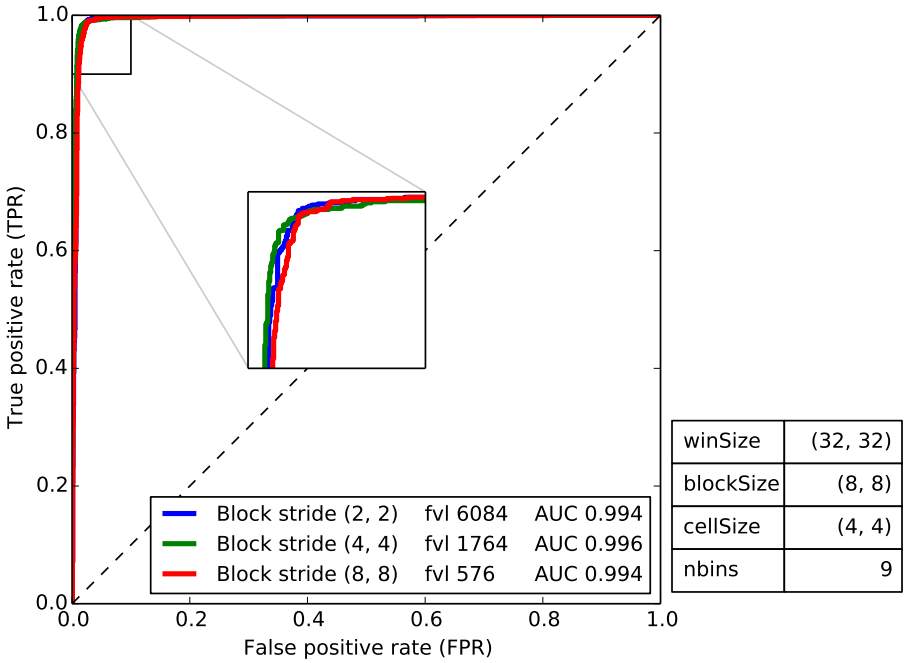

Figure 6.3: ROC curves for different block strides.

Width and height of cell size (2,2) correspond to a quarter of those of the used block size. Cell sizes (4,4) and (8,8) yield lower AUCs of 0.994 and 0.992. Reducing cell size from (4,4) to (2,2) improves AUC only by 0.001 but at the same time the length of feature vector is quadrupled. In other words, cell size (2,2) provides slightly better performance than cell size (2,2) with the cost being increased computation time.

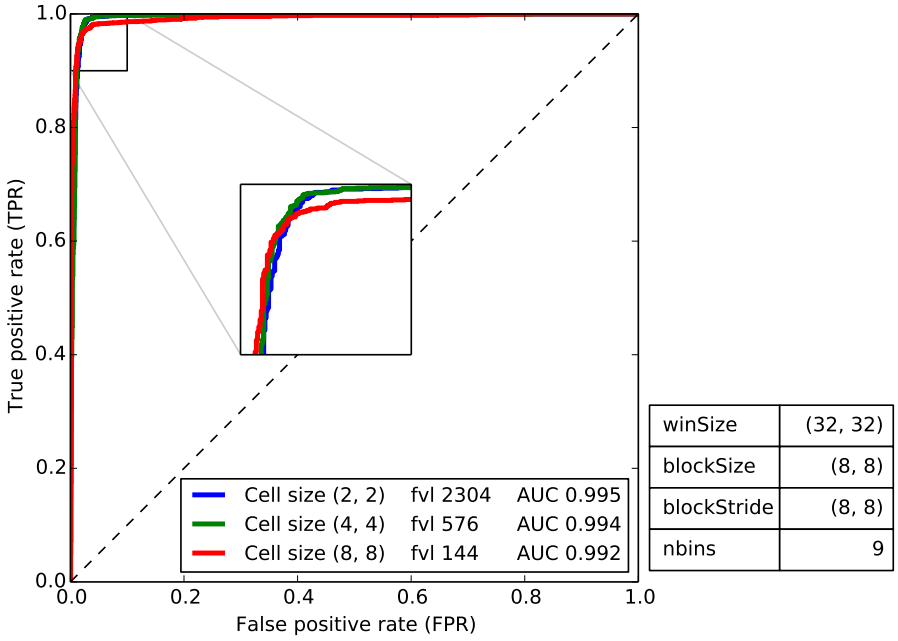

Figure 6.4: ROC curves for different cell sizes.

Increasing the number of orientation bins also improves SVM performance, as the ROC curves in Figure 6.5 indicate. Using only 1 bin gives poor AUC of 0.696. When only 1 bin is used, the difference between samples is determined by the sum of magnitudes in a sample. Increasing the number of orientation bins from 1 to 2 leads to 0.266 increase in AUC to 0.962. 3 orientation bins yield AUC of 0.986. When the number of bins is doubled to 6, AUC increases by 0.005 to 0.991. After this point, the increase of AUC slows down because 9 orientation bins raise AUC by 0.004 to 0.995. When the number of bins is doubled again to 18, 0.003 increase in AUC is seen, leading to a value of 0.998. Almost perfect classification with AUC of 0.999 is obtained when the nbins parameter value of 36 is used.

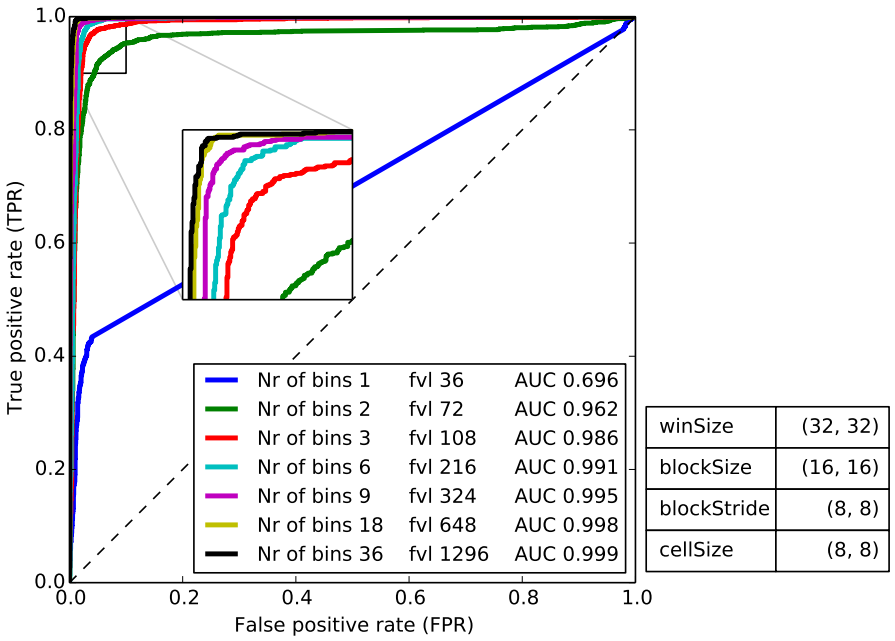

Figure 6.5: ROC curves for different number of orientation bins.

Another way of inspecting the difficulty of the classification problem is to visualize the separability of the classes. Figure 6.6 shows a histogram of 2000 positive and 2000 "easy" negative testing data point projections onto normal vector **w** of linear SVM. The projection can be simply done by calculating dot product with features and SVM weights. Location 0 on the x-axis of the histogram represents the location of the separating hyperplane. The histogram indicates that the classes are well separable even though both classes consist of a small portion of examples which somehow remind the opposite class. Those examples are identified by the overlapping tails of the histograms.

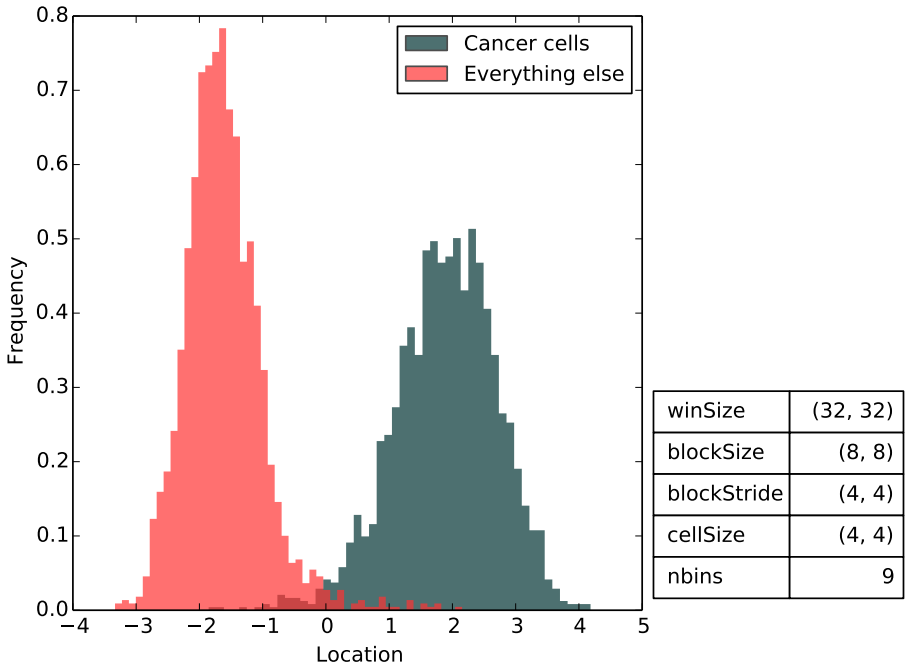

Figure 6.6: Visualization of class separability of unseen training data points.

6.1.3 Effect of Finding Hard Examples

Figure 6.7 shows how AUC of SVM decreases on the left-hand side and how detection accuracy increases in terms of averaged F_1-score on the right-hand side during iterative training. The ideal `detectMultiScale` parameter values were not used in this test, which is why the resulted F_1-scores are not the main point here so much as the general increase in those values is. The models on each iteration have been trained with 75 % of random positive examples and 75 % of random negative examples. The ROC curves have been calculated by classifying the remaining 25 % of each class examples. On each iteration parameter C of SVM was cross-validated on the range of $10^{-2}, \ldots, 10^{2}$, resulting every time to selecting $C = 0.1$.

On the first iteration with all the positive and all the "easy" negative examples AUC reaches a very high value of 0.997. However, SVM does not perform as well as AUC might suggest when the model classifies images during the second iteration in sliding window method that is run on the same input images from where the training examples of that particular model were cropped, leading to poor F_1-score of 0.11 and 1064 hard examples. These hard examples lower AUC by 0.004 to 0.993. Employing the large amount of initial hard examples when training a new model for the third iteration, causes F_1-score to increase to 0.47. Also, the number of hard examples decreases on the same iteration to 295, which corresponds to roughly one third of their amount on the previous (second) iteration. As the iterations progress, the rate of finding less hard examples slows down along with the rate of increase in

F_1-score. After 15 iterations, the iterative training is finished because the number of hard examples has decreased to 50, which we consider to be low enough amount corresponding to 5 % of the initial number of hard examples. On the last iteration F_1-score reaches value of 0.77 and AUC has lowered to 0.982.

One might easily misunderstand the ROC curves in Figure 6.7. It is true that classification accuracy lowers in terms of AUC as the iterations proceed, but what is also happening is that the level of generalization of the model is getting higher and higher on each iteration as the model is trained with more and more hard examples. The semi-automatically gathered training examples are too easy to separate, which cause SVM to overfit the training data and to generalize poorly to unseen data if no hard examples are mined. Thus, the ROC curve on the last iteration correspond better to the reality of the difficulty of the classification problem than the one drawn on the first iteration.

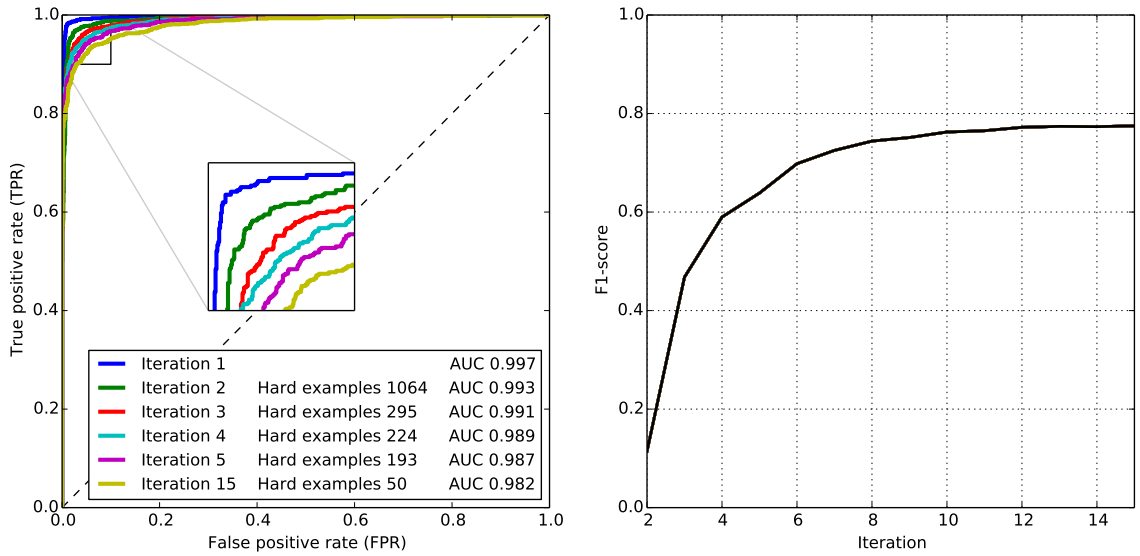

Figure 6.7: The results of iterative SVM training. Left: classification performance demonstrated by ROC curves for each SVM training iteration. Right: detection performance demonstrated by averaged F_1-score over each training image from days 1-3 as a function of iterations.

6.2 Overall Performance

This section presents the results of studying the overall performance of the framework in the testing phase. Subsection 6.2.1 answers to the question: "What sliding window parameter values should be chosen for cell detection?" and Subsection 6.2.2 answers to the question: "How accurate is the estimated growth curve?".

6.2.1 Sliding Window Method Parameters

This subsection shows the results of performing a grid search on the scale, nlevels, winStride and finalThreshold parameters of `detectMultiScale` in terms of F_1-score and computation time. The aim is therefore to find and justify values that can be suggested for sliding window method in cell detection with HOG.

Our original aim was to evaluate the sliding window method with ROC curves, which unfortunately appeared to be impossible after many failed attempts. It was impossible to calculate complete ROC curves with the sliding window method because TPR did not ever reach its maximum value of 1.0 even though the threshold of SVM was lowered beyond the point where each example should have been considered as TP. We suspect that the cause for this was the algorithm in OpenCV which merged too many nearby detections as one detection because the annotations that never got detected were those which overlapped a lot with other annotations. That is why we use F_1-score when assessing sliding window method parameters.

We found that resizing window size to various scales in sliding window procedure only lowers performance from F_1-score of 0.85 to F_1-score of 0.78, which is demonstrated on the left-hand side in Figure 6.8. Multi-scale search enables detection of large cells which, however, exist in small numbers. Sizes of annotated cancer cells vary in our images within a fairly narrow range, as presented in Section 5.2. That is why we suggest performing the sliding window procedure on single-scale using window size of 32×32, which has aspect ratio and size close to the average of those in annotated cells. Another advantage with single-scale detection is the shortest possible computation time, 1.10 seconds.

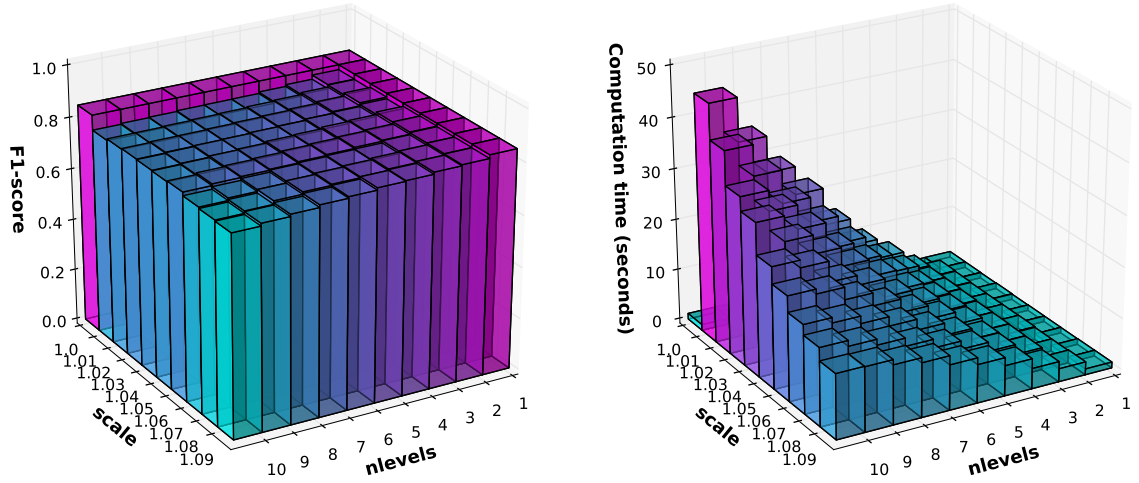

Figure 6.8: 3-D bar graph visualization of grid search performed on scale and nlevels parameters of `detectMultiScale` function. On the left-hand side there are F_1-scores on the z-axis averaged over all the test images and on the right-hand side there are corresponding computation times.

Figure 6.9 demonstrates that using a default finalThreshold value of 2.0 yields the highest averaged F_1-score of 0.85 together with using winStride parameter value of (4,4), which has width and height corresponding to half of those of utilized blockSize. Those values have also the shortest computation time of 3.26 seconds when compared to those of other combinations of the parameter values. Smaller parameter values lead to lower cell detection accuracy and increased computation time.

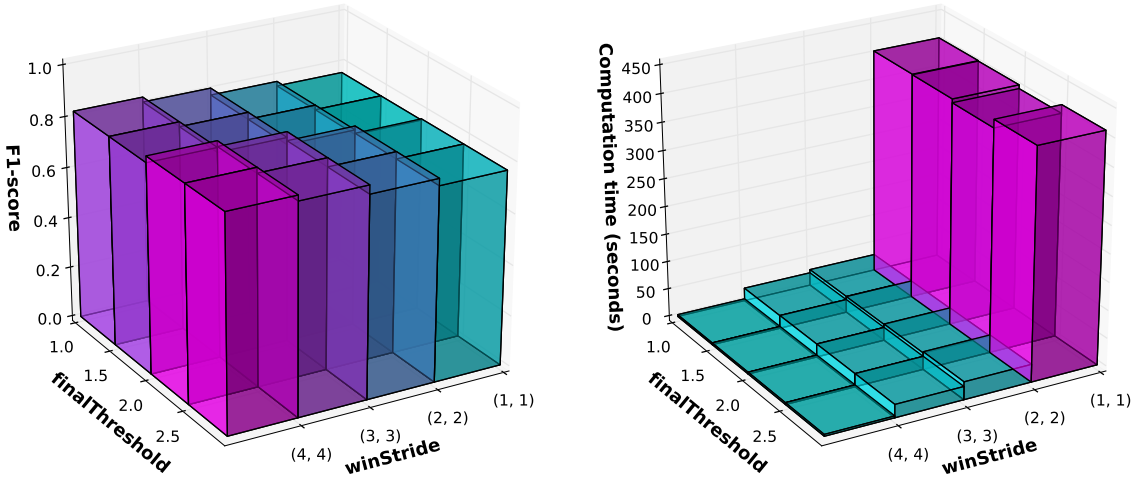

Figure 6.9: 3-D bar graph visualization of grid search performed on finalThreshold and winStride parameters of `detectMultiScale` function. On the left-hand side there are F_1-scores on the z-axis averaged over all the test images and on the right-hand side there are corresponding computation times.

Figure 6.10 presents the effect that hitThreshold has on F_1-score. HitThreshold parameter is the same as the bias term of SVM which corresponds to the SVM hyperplane's distance from the origin, thus regulating the sensitivity of the model. The figure shows that images of each day prefer slightly different hitThreshold values in terms of F_1-score. Day 1-3 images reach their highest F_1-scores of 0.80, 0.84 and 0.88 with more "conservative" classifiers that other day images prefer, having higher hitThresholds 1.02, 0.94 and 0.86. Day 4 and 6 prefer more "liberal" models because their best F_1-scores of 0.85 and 0.82 were achieved with lower hitThresholds 0.41 and 0.40. Day 5 images gain their highest averaged F_1-score of 0.87 with hitThreshold value of 0.68, which is near the average hitThreshold of all images, 0.72. The average of the best F_1-scores of each test image while the threshold was varied is 0.85.

The results that Figure 6.10 presents, suggest that the default hitThreshold value of 0.00 does not give the optimal cell detection outcome. Hence, its neighboring values should be examined. Furthermore, it seems that when selecting the most suitable hitThreshold value, a trade-off has to be made between cell detection accuracy being higher on some day images than others.

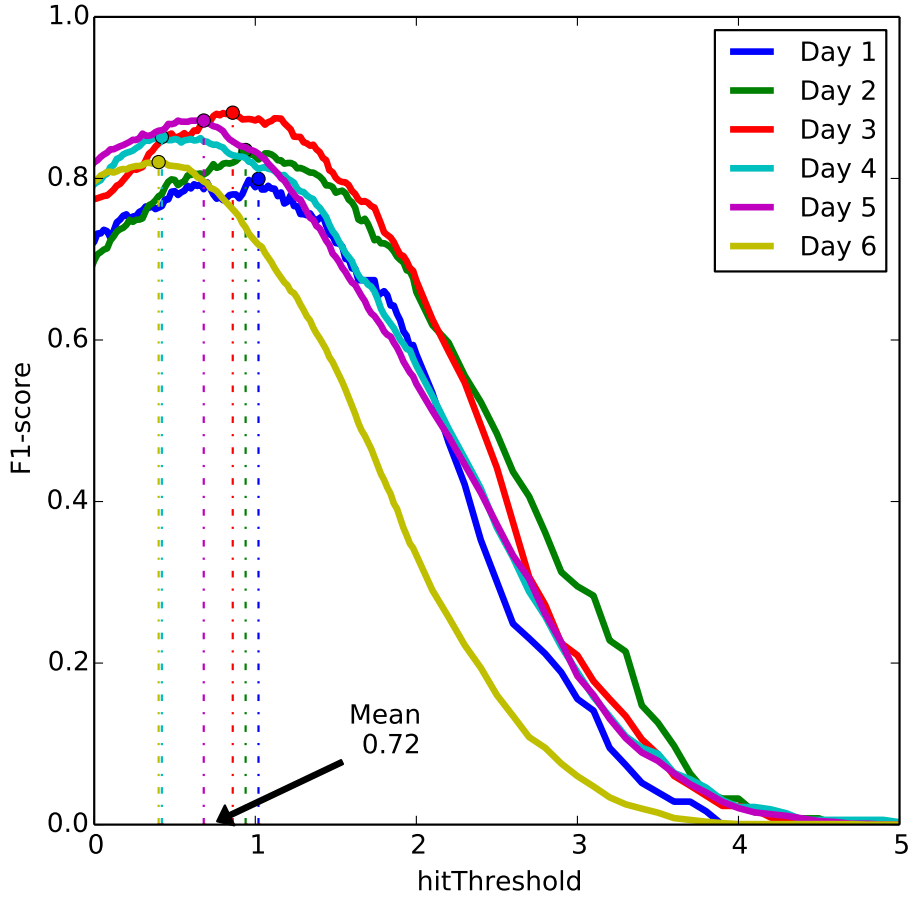

Figure 6.10: Averaged F_1-score for each day images as a function of hitThreshold parameter values. Highest F_1-score of each curve is indicated by a stem plot.

6.2.2 Growth Curve Estimation

This subsection is concentrated on reporting the results of growth curve estimation, which is the final step in the proposed cell detection framework. Figure 6.11 shows estimated growth curves on the left-hand side and scatter plots of their BSI measures on the right-hand side, presented as TET vs. TEE. `HOGDescriptor` and `detectMultiScale` parameter values were selected based on all the results that are presented in this chapter. The growth curves were generated using hitThresholds of 0.6 (upper row) and 0.7. It should be noted that the results in Figure 6.11 are slightly biased when it comes to the sensitivity of SVM because the hitThreshold values were selected based on Figure 6.10, which was calculated using the same images as in Figure 6.11. In a real-life application that cannot be done because testing data labels are unknown. The hitThreshold value should be inferred using training data and hope that the selected value works also well with unseen images.

The Figure 6.11 shows that both of the estimated growth curves follow the manual counts in a favorable manner. The both estimated curves also overestimate the number of cells on day 1-3 images and underestimate the number of cells on day 6

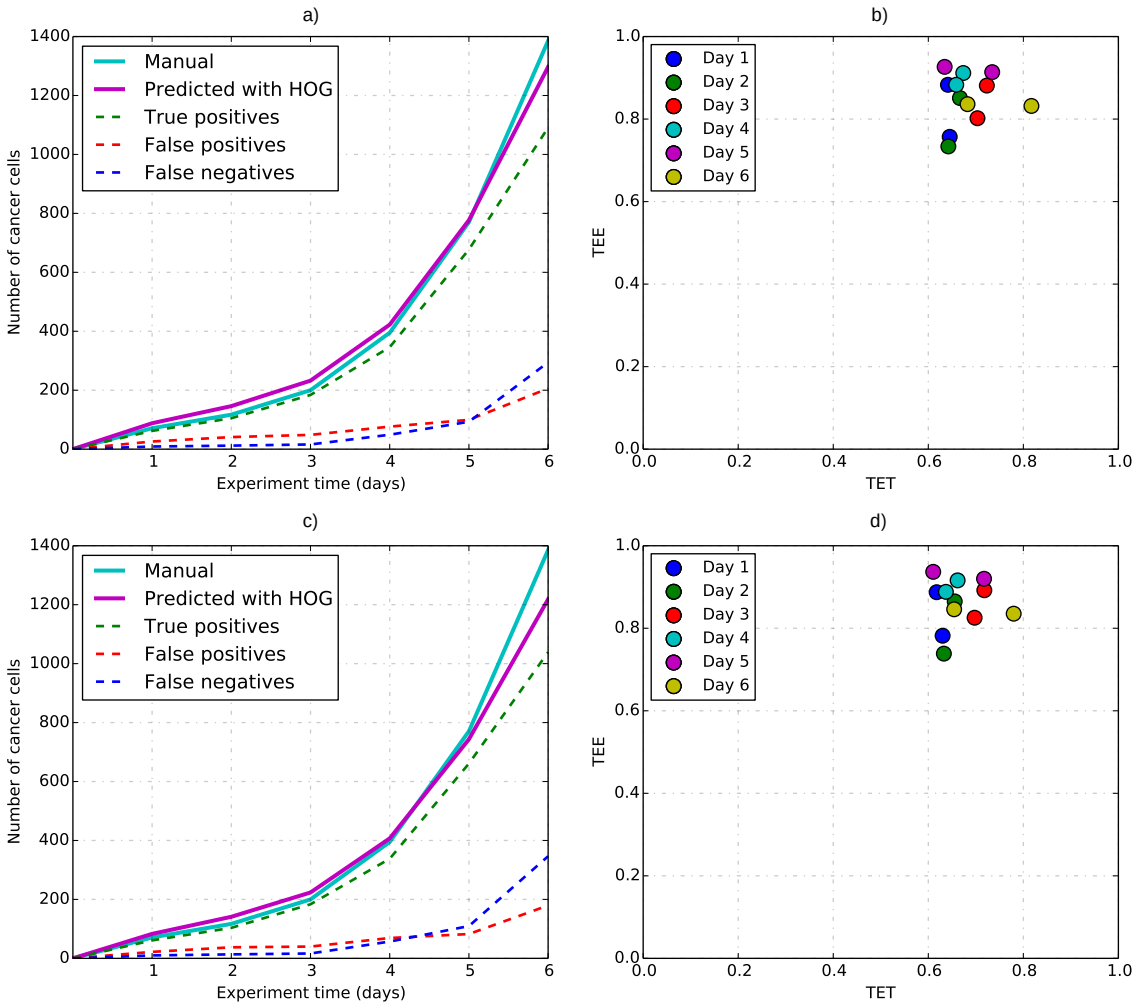

Figure 6.11: Estimated growth curves and scatter plots of their BSI measures TET and TEE, using two different hitThreshold values. The results in each row are from the same test, so that results in the first row used hitThreshold of 0.6 and the results in the second row used hitThreshold of 0.7.

images. More "liberal" SVM classifier with hitThreshold of 0.6 predicts the number of cells closer to the reality on day 5 and 6 images than the more more "conservative" classifier with hitThreshold 0.7. The conservative classifier is more accurate with day 1-4 images than the liberal classifier.

When looking at the manual counts, the number of cells grows almost linearly from 70 to 200 on during days 1-3. After that, the number of cells roughly doubles when measured on days 4 and 5. In the end, the cell quantity grows from ~800 to ~1400 when moved on from day 5 to day 6.

In real-life applications, it is almost impossible to avoid FPs, and thus more attention should be given to the total number of detections, TP+FP, instead of only looking at the correct amount of TPs. It seems that the most suitable SVM thresholds for the images of each day could be the ones which produce equal amounts

of FPs and FNs, which is exactly what happens on day 5 in the upper left corner of Figure 6.11. In other words, with such hitThreshold value the estimated counts would meet the manual counts perfectly, because then the amount of detected FPs closes the gap to the manual counts that the amount of TPs leaves underestimated.

The BSI graphs of Figure 6.11 support the argument of accurate cell detection with HOG because TET and TEE are both large in both BSI graphs. TET values are centered around 0.7 and TEE values are centered around 0.9. The segmentation distance (4.11) is 0.35 with the liberal classifier and 0.36 with the conservative classifier.

Figure 6.12 shows cropped day 5 image of the result of cell detection with HOG features. The figure demonstrates effective detection of cancer cells and the difficulty of the problem.

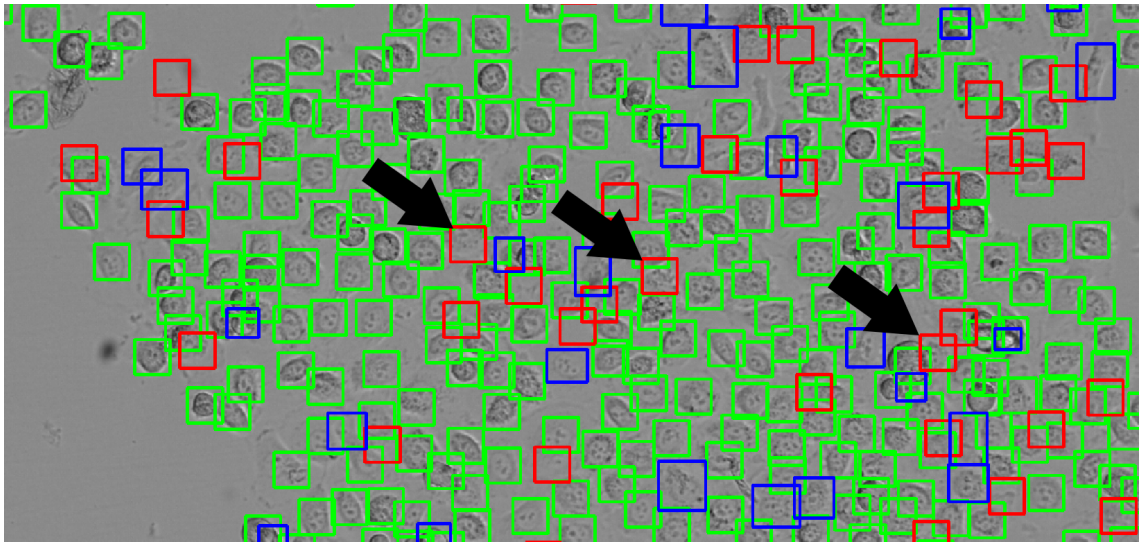

Figure 6.12: Cropped day 5 image of the result of cell detection with HOG features. The green rectangles denote true positives, the red rectangles denote false positives and the blue rectangles denote false negatives. The black arrows denote three false negatives where the cell detection framework has correctly detected a cell that the human annotator has missed when annotating.

7. CONCLUSIONS

The results of this thesis imply that Histogram of Oriented Gradients feature descriptors can be successfully applied to cell detection from bright-field microscope images. Growth curve, which agrees favorably with manual counts can be estimated by counting the occurrence of the detected cells from microscope images taken on subsequent days. Automated algorithm counted the cells in a more objective, consistent and faster manner than manual counting.

The proposed cell detection framework learns a Support Vector Machine model iteratively by finding hard examples. The iterative training process was noticed to be a crucial step for eliminating false positive detections, producing cross-validated ROC AUC of 0.98. In the testing phase, two other performance evaluation metrics were used. A segmentation distance of 0.35 according to Bivariate Similarity Index is acquired. When SVM threshold is varied for each image in the testing phase, F_1-score averaged over the peak F_1-scores of each image reaches value of 0.85.

As a result of thorough investigation of HOG parameters with ROC curves in the training phase, the parameter values shown in Table 7.1 are suggested for HOG descriptor in cell detection. These values serve as a reasonable starting point and the accuracy of the descriptor can be improved by increasing the number of orientation bins or by minimizing the cell size to 2×2.

Table 7.1: Suggested HOG descriptor parameter values for cell detection.

Parameter	Value
Descriptor window size	(32, 32)
R-HOG block size	(8, 8)
Block step size	(4, 4)
Cell size	(4, 4)
Number of orientation bins	9
Maximum number of detection window increases	1

The results emphasize the importance of clean imaging conditions. The hardest examples consisted mostly of blurred spots that were caused by unclean camera lens. Those spots were detected easily as cells because their shape and size were very similar to those of many of the cells, especially the ones out of focus.

As suggestions for future research, it could be interesting to find out how deep learning works with cell detection or to investigate how different SVM kernels perform. Furthermore, it would be worth studying if a stack of different focus levels of microscope images could improve the accuracy of the system when compared to the current situation, where single focus level is used. Even though bright-field microscope images of cancer cells were used in this thesis, we believe the detection framework could perform successfully also with other kinds of microscopy techniques and cell types, if the training is done with according material. As a final development proposal, it would be helpful if the cell detection framework had a graphical user interface, where one could for example change manually the sensitivity of SVM.

When it comes to suggested improvements, there is room for improvement in OpenCV. Its documentation is practically nonexistent, which lead to certain drawbacks in the implementation. Also, nlevels parameter is placed wrongly as `HOGDescriptor` parameter as it should belong as part of `detectMultiScale` parameters. Current placement of the parameter is not logical and makes the task of varying its values unnecessarily complicated.

Another outcome of this thesis is that the software implementation and the data set will be publicly available and can be downloaded from the supplementary website[1]. Also, a research paper summarizing the essence of the results of this thesis is submitted to the IEEE International Symposium on Biomedical Imaging (ISBI 2015) [47].

In summary, it is possible to implement a complete and robust cell detection framework with HOG features, which yields accurate results with relatively low error rate. If the aim is to estimate growth curve, perhaps the most important question lies in selecting the correct SVM threshold. The ideal threshold vary from images to images, depending on the number of cells in the image.

[1] `http://code.google.com/p/hog-cell-detection-framework/`

REFERENCES

[1] Lodish H., Berk A., Kaiser C., Kreiger M., Bretscher A., Ploegh H., Amon A. & Scott M. Molecular Cell Biology. 7th edition. New York, NY, USA, 2012, Freeman and Co. 1154 p.

[2] Dang C., Gilewski T.A., Surbone A. & Norton L. Growth Curve Analysis. In: Kufe D.W., Pollock R.E., Weichselbaum R.R., Bast R.C., Gansler T.S., Holland J.F. & Frei E. (eds.). Holland-Frei Cancer Medicine. 6th edition. Hamilton, ON, Canada, 2003, B.C. Decker Inc. 2699 p.

[3] Peng H. Bioimage Informatics: A New Area of Engineering Biology. Bioinformatics 24(2008)17, pp. 1827-1836.

[4] Prewitt J.M.S. & Mendelsohn M.L. The Analysis of Cell Images. Annals of the New York Academy of Sciences 128(1966)3, pp. 1035-1053.

[5] Moore G.E. Cramming More Components onto Integrated Circuits. Electronics 38(1965)8, pp. 114-117.

[6] Sjostrom P.J., Frydel B.R. & Wahlberg L.U. Artificial Neural Network-Aided Image Analysis System for Cell Counting. Cytometry 36(1999)1, pp. 18-26.

[7] Dalal N. & Triggs B. Histograms of Oriented Gradients for Human Detection. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 20-26 June, 2005. IEEE, Vol. 1, pp. 886-893.

[8] Déniz O., Bueno G., Salido J. & De la Torre F. Face Recognition using Histograms of Oriented Gradients. Pattern Recognition Letters 32(2011)12, pp. 1598-1603.

[9] Rybski P.E., Huber D., Morris D.D., & Hoffman R. Visual Classification of Coarse Vehicle Orientation using Histogram of Oriented Gradients Features. In Proceedings of the IEEE Intelligent Vehicles Symposium (IV), San Diego, CA, USA, 21-24 June, 2010. pp. 921-928.

[10] Lowe D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60(2004)2, pp. 91-110.

[11] Felzenszwalb P.F., Girshick R.B., McAllester D. & Ramanan D. Object Detection with Discriminatively Trained Part-Based Models. IEEE Transactions on Pattern Analysis and Machine Intelligence. 32(2010)9, pp. 1627-1645.

[12] Barbu T. SVM-based Human Cell Detection Technique using Histograms of Oriented Gradients. In Proceedings of the 17th WSEAS International Conference on Applied Mathematics (AMATH '12), Montreux, Switzerland, 29-31 December, 2012, pp. 156-160.

[13] Otsu N. A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics 9(1979)1, pp. 62-66.

[14] Malpica N., Ortiz de Solórzano C., Vaquero J.J., Santos A., Vallcorba I., García-Sagredo J.M. & Pozo F.D. Applying Watershed Algorithms to the Segmentation of Clustered Nuclei. Cytometry 28(1997)4, pp. 289-297.

[15] Lin G., Adiga U., Olson K., Guzowski J.F., Barnes C.A. & Roysam B. A Hybrid 3D Watershed Algorithm Incorporating Gradient Cues and Object Models for Automatic Segmentation of Nuclei in Confocal Image Stacks. Cytometry Part A 56(2003)1, pp. 23-36.

[16] Megason S.G. & Fraser S.E. Imaging in Systems Biology. Cell 130(2007)5, pp. 784-795.

[17] Lu C. & Tang X. Surpassing Human-Level Face Verification Performance on LFW with GaussianFace. Hong Kong 2014, Department of Information Engineering, The Chinese University of Hong Kong. Technical report. 13 p.

[18] Krizhevsky A., Sutskever I. & Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the Neural Information Processing Systems (NIPS'12), Lake Tahoe, Nevada, USA, 3-6 December, 2012, pp. 1097-1105.

[19] Taigman Y., Yang M., Ranzato M.A. & Wolf L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Columbus, Ohio, USA, 24-27 June, 2014, pp. 1701-1708.

[20] Bishop C.M. Pattern Recognition and Machine Learning. New York, NY, USA, 2006, Springer. 738 p.

[21] Duda R.O., Hart P.E. & Stork D.G. Pattern Classification. 2nd edition. New York, NY, USA, 2001, John Wiley & Sons. 654 p.

[22] Tibshirani R. Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological) 58(1996)1, pp. 267-288.

[23] Amit Y. & Felzenszwalb P. Object Detection, in: Katsushi I. (ed.), Computer Vision: A Reference Guide, New York, NY, USA, 2014, Springer. pp. 537-542.

[24] Ng A.Y. & Jordan M.I. On Discriminative vs. Generative Classifiers: A comparison of Logistic Regression and Naive Bayes. In: Diettrich T.G., Becker S. & Ghahramani Z. (eds.). Advances in Neural Information Processing Systems 14. In Proceedings of the 2001 Conference, Cambridge, Mass, USA, 2002, MIT Press. pp. 841-848.

[25] Blumer A., Ehrenfeucht A., Haussler D. & Warmuth M.K. Occam's Razor. Information Processing Letters 24(1987)6, pp. 377-380.

[26] Myatt, G.J. Making Sense of Data: A Practical Guide to Exploratory Data Analysis and Data Mining. Hoboken, NJ, USA, 2007, John Wiley & Sons. 288 p.

[27] Gonzalez R.C. & Woods R.E. Digital Image Processing. 2nd edition. Upper Saddle River, NJ, USA, 2002, Prentice Hall. 793 p.

[28] Porikli F. 2005. Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 20-26 June, 2005. IEEE, Vol. 1, pp. 829-836.

[29] Viola J. & Jones M. Rapid Object Detection using a Boosted Cascade of Simple Features. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii, USA, 8-14 December, 2001. IEEE, Vol. 1, pp. 511-518.

[30] Cortes C. & Vapnik V. Support-Vector Networks. Machine Learning 20(1995)3, pp. 273-297.

[31] Wu X., Kumar V., Quinlan J.R., Ghosh J., Yang Q., Motoda H., McLachlan G.J., Ng A., Liu B., Yu P.S., Zhou Z.-H., Steinbach M., Hand D.J., Steinberg D. Top 10 Algorithms in Data Mining. Knowledge and Information Systems 14(2008)1, pp. 1-37.

[32] Burges J.C.C. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2(1998)2, pp. 121-167.

[33] Chang C.C. & Lin C.J. Training ν-Support Vector Classifiers: Theory and Algorithms. Neural Computation 13(2001)9, pp. 2119-2147.

[34] Schölkopf B., Smola A.J., Williamson R.C. & Bartlett P.L. New Support Vector Algorithms. Neural computation 12(2000)5, pp. 1207-1245.

[35] Hsu C.-W., Chang C.-C., & Lin C.-J. 2003. A Practical Guide to Support Vector Classification. Taiwan 2003, Department of Computer Science, National Taiwan University. Technical report. 16 p.

[36] Everingham M., Zisserman A., Williams C., Van Gool L., Allan M., Bishop C., Chapelle O., Dalal N., Deselaers T., Dorko G., Duffner S., Eichhorn J., Farquhar J., Fritz M., Garcia C., Griffiths T., Jurie F., Keysers D., Koskela M., Laaksonen J., Larlus D., Leibe B., Meng H., Ney H., Schiele B., Schmid C., Seemann E., Shawe-Taylor J., Storkey A., Szedmak S., Triggs B., Ulusoy I., Viitaniemi V. & Zhang J. The 2005 PASCAL Visual Object Classes Challenge. In: Quiñonero-Candela J., Dagan I., Magnini B. & d'Alché-Buc F. (eds.). Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment (First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, 11-13 April, 2005). Lecture Notes in Computer Science. Vol. 3944. Springer, pp. 117-176.

[37] Fawcett T. An Introduction to ROC Analysis. Pattern Recognition Letters 27(2006)8, pp. 861-874.

[38] Gorunescu F. Data Mining: Concepts, Models and Techniques. Cluj-Napoca, Romania, 2011, Springer. 360 p.

[39] Chinchor N., MUC-4 Evaluation Metrics. Fourth Message Understanding Conference (MUC-4): In Proceedings of a Conference Held in McLean, Virginia, June 16-18, 1992. Morgan Kaufmann Publishers, pp. 22-29.

[40] Jaccard P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 37(1901)142, pp. 547-579.

[41] Choi S.-S., Cha S.-H. & Tappert C. C. A Survey of Binary Similarity and Distance Measures. International Journal of Systemics, Cybernetics, and Informatics 8(2010)1. pp. 43-48.

[42] Dima A.A., Elliott J.T., Filliben J.J., Halter M., Peskin A., Bernal J., Kociolek M., Brady M.C., Tang H.C. & Plant A.L. Comparison of Segmentation Algorithms For Fluorescence Microscopy Images of Cells. Cytometry Part A 79(2011)7, pp. 545-559.

[43] OpenCV [WWW]. [accessed on 17th September 2014].
Available at: `http://opencv.org/`.

[44] NumPy [WWW]. [accessed on 17th September 2014].
Available at: `http://www.numpy.org/`.

[45] scikit-learn [WWW]. [accessed on 17th September 2014].
Available at: http://scikit-learn.org/.

[46] LIBSVM [WWW]. [accessed on 23rd September 2014].
Available at: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[47] Tikkanen T., Ruusuvuori P., Latonen L. & Huttunen H. Training Based Cell Detection from Bright-field Microscope Images. IEEE International Symposium on Biomedical Imaging (ISBI) (Submitted).